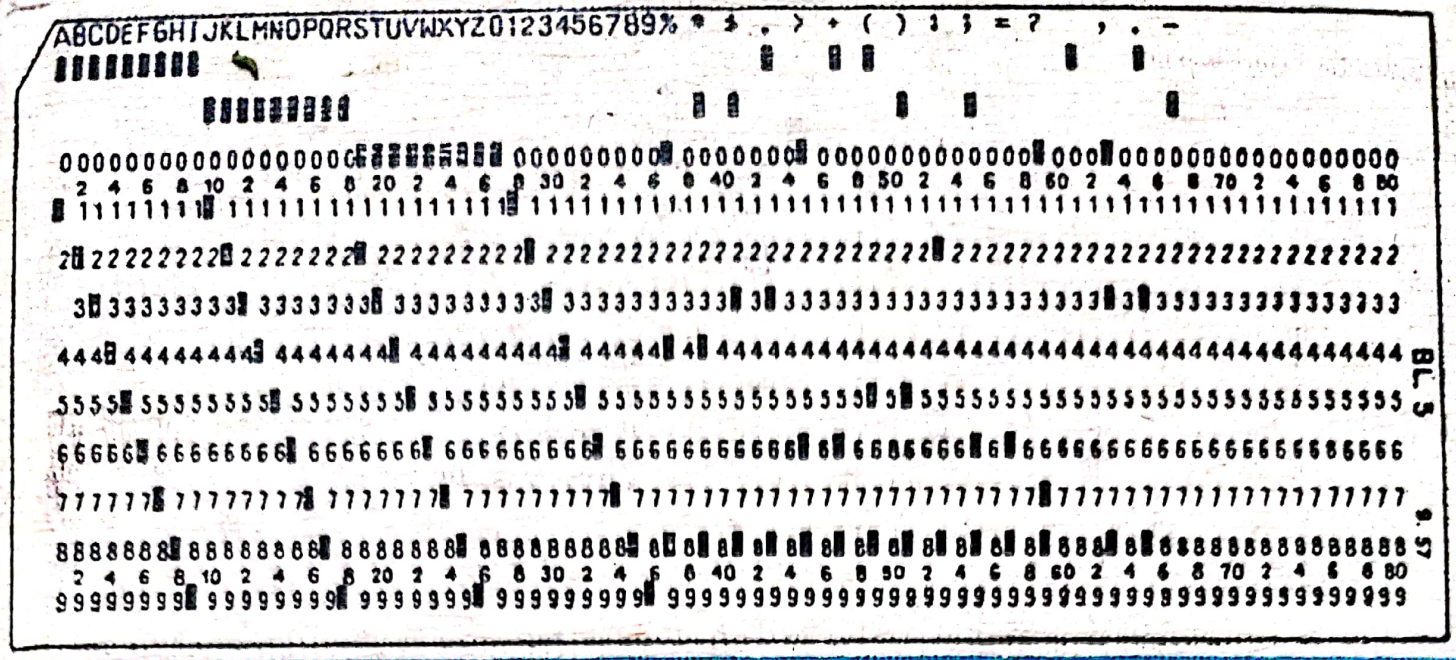


ION VĂDUVA NICOLAE POPOVICIU

INTRODUCERE ÎN PROGRAMAREA AUTOMATĂ

MINISTERUL EDUCAȚIEI ȘI ÎNVĂȚĂMÎNTULUI

ION VĂDUVA NICOLAE POPOVICIU

INTRODUCERE ÎN PROGRAMAREA AUTOMATĂ

cu aplicații la cercetarea științifică

Editura didactică și pedagogică
București 1973

Scanned with OKEN Scanner

Prefață

Lucrarea are ca scop să transmită celor interesați cunoștințele de bază legate de prelucrarea automată a datelor și folosirea unui calculator pentru rezolvarea problemelor de cercetare științifică, urmărindu-se: elaborarea modelului matematic al unei probleme, alegerea unui algoritm de rezolvare numerică, întocmirea schemei logice de calcul corespunzătoare algoritmului, elaborarea programului într-un limbaj de programare automată, interpretarea și folosirea rezultatelor.

Cele cinci capitole ale cărții își propun atingerea acestui scop. Primul capitol se ocupă de structura — fizică și logică — a unui sistem de calcul electronic, punîndu-se în evidență principiile de funcționare și posibilitățile sale de lucru.

Capitolul al doilea tratează aspecte legate de construirea schemelor logice de calcul.

În capitolul al treilea este prezentată o introducere în limbajul FORTRAN, care este unul din limbajele simple, destinat rezolvării problemelor de cercetare științifică.

Prezentarea limbajului este însoțită de exemple și de exerciții, ale căror soluții sînt date la sfîrșitul capitolului.

Capitolul al patrulea este dedicat problemelor privind utilizarea memoriilor externe (discuri și benzi magnetice) ale unui sistem de calcul.

Capitolul al cincilea expune unele probleme rezolvate, insistînd mai mult asupra aspectelor legate de construirea algoritmilor și programarea acestora. O secțiune a acestui capitol se ocupă de folosirea unor programe dintr-o bibliotecă de subprograme, destinată rezolvării problemelor de programare liniară (§ 5.1).

Scanned with OKEN Scanner

Problemele prezentate în acest capitol exploatează, într-o ordine logică, multe din cunoștințele conținute în capitolele anterioare. Exemplele tratate în diferitele capitole ale cărții scot în evidență unele dificultăți care pot apare în problemele concrete și indică moduri de depășire a acestora.

Cartea este scrisă riguros, dar la un nivel mediu, evitîndu-se multe din demonstrațiile prea complicate. Lămurirea problemelor mai dificile se face prin exemple, preferîndu-se explicarea prin mijloace intuitive celei bazate pe demonstrații anevoioase. Acest mod de tratare a fost abordat spre a face expunerea cît mai accesibilă și cu un folos imediat pentru cel care o citește.

Toate chestiunile prezentate au trimiteri la bibliografie, care a fost alcătuită separat, la sfîrșitul fiecărui capitol.

Cartea se adresează unor cititori de profesii diferite, ca matematicieni, ingineri, fizicieni, economiști, sociologi, medici, etc. Desigur, ea poate fi înțeleasă cu ușurință de cei care au unele cunoștințe atît în domeniul calculatoarelor cît și al capitolelor de matematică ce intervin în expunere. Capitolele I, III și IV pot fi înțelese ușor și fără astfel de cunoștințe prealabile, însă cititorul progresează rapid, mai ales dacă susține studiul prin practică la un calculator. Cartea poate fi, de asemenea, de un real folos studenților care se inițiază în programarea calculatoarelor.

Capitolele cărții pot fi abordate și separat (mai puțin ultimul care este o sinteză aplicativă a celorlalte), iar exemplele tratate sperăm că vor putea fi utilizate la multe din problemele concrete care îl interesează pe cititor, sau măcar vor constitui puncte de sprijin în acest sens.

AUTORII

Scanned with OKEN Scanner

Capitolul I

INTRODUCERE ÎN CUNOAȘTEREA UNUI SISTEM DE CALCUL

1.1. INTRODUCERE

Calculatoarele electronice au devenit instrumente de lucru foarte importante în mîna omului. Exploatarea lor este însă în funcție de pregătirea utilizatorului. Cartea de față are ca scop inițierea și transmiterea unui volum de cunoștințe destul de bogat, legate de utilizarea unui calculator și de posibilitățile de lucru ale acestuia.

Pentru folosirea calculatorului în problemele de cercetare științifică, utilizatorul, care poate fi un cercetător din orice domeniu de activitate, trebuie să aibă la îndemînă modelul matematic al problemei sale și algoritmul de rezolvare al acestei probleme. Aceste elemente sînt de o foarte mare importanță, întrucît orice calculator poate rezolva o problemă numai dacă aceasta îi este transmisă într-o anume formă. Forma de transmitere a problemei trebuie privită sub două aspecte: unul este legat de tipul concret de calculator folosit, care cere de exemplu trecerea problemei pe banda perforată sau pe cartele perforate, etc., iar al doilea aspect, independent de tipul de calculator utilizat, este legat de algoritmul de rezolvare al problemei. Algoritmul trebuie să realizeze reducerea problemei la operațiile pe care poate să le execute un calculator și anume: operații aritmetice, operații de comparare, operații logice, operații de citire și depunere pe un mediu exterior. De exemplu, rezolvarea unei ecuații diferențiale se face prin reducerea problemei la a calcula funcția $y(x)$ numai într-un număr finit de puncte, iar calculul unei integrale definite se poate face cu ajutorul calculatorului numai prin folosirea unei formule de integrare numerică.

Analiza numerică permite să se aproximeze, cu o aproximație oricît de bună, soluția exactă, de multe ori folosindu-se numai cele patru operații aritmetice, dar cu prețul unui număr mare de astfel de operații simple.

Specificăm de la început că, pe tot parcursul cărții, ne referim numai la calculatoarele electronice digitale, care sînt calculatoare universale, și nu și la calculatoarele analogice, care sînt calculatoare specializate în rezolvarea unui anumit tip de problemă.

Cititorul trebuie să se familiarizeze cu ce pot face și ce nu pot face calculatoarele, cu principiile de bază ale funcționării lor, dar mai

Scanned with OKEN Scanner

ales cu utilizarea lor. Ideile de bază nu sînt dificile dacă sînt prezentate într-o ordine ierarhică, iar intenţia acestei cărţi este tocmai de a face acest lucru.

În titlul acestui capitol am folosit noţiunea de *sistem de calcul*, iar pînă acum ne-am referit tot timpul la *calculatoare*. Motivul este acela că un sistem de calcul are cel puţin un calculator, care formează partea cea mai importantă a sa, la care se adaugă echipamentul periferic foarte diversificat. Adeseori însă, în vorbirea curentă, prin calculator se înţelege întregul sistem de calcul. Se spune, de exemplu, „sala calculatorului" nu „sala sistemului de calcul", (calculatorul propriu-zis este *unitatea centrală* a sistemului; vezi § 1.2).

Pînă în prezent se cunosc numeroase sisteme de calcul, fiecare sistem avînd unul sau mai multe tipuri. Menţionăm cîteva sisteme, împreună cu tipurile lor în paranteze, fără ca ordinea de enumerare să însemne o ierarhizare: IBM (1401, 1411, 7030, 7040, 7044, 7080, 7090, 7094, STRETCH, 360, 370), HONEYWELL (200, 400, 800, 1800, 1400, 2200), UNIVAC (80, 90, 1004, 1050, 1107, 1108, 491, 492, 9300), BARROUGHS (220, 300, 5500), CDC (3100, 3200, 3600, 6600), RCA (301, 501, 601, 3301), NCR (310, 315), GENERAL ELECTRIC-215, ELLIOTT (803, 4100), ENGLISH-ELECTRIC, ICT-1900, BULL GAMMA-60, SIEMENS (303, 3003, 4004), IRIS-50, FELIX C-256, OLIVETTI, NIPON ELECTRIC, etc. Pentru detalii se poate consulta [5] şi [11].

Unele dintre sistemele de un anumit tip de mai sus formează o *serie* sau o *familie*.

Caracteristicile seriei sînt date de caracteristicile unităţii centrale folosite. Unitatea centrală indică modelul sistemului. De exemplu, seria ICT 1900 are modelele 1902, ..., 1909, etc; seria IBM 360 are modelele 20, 30, 40, 44, 50, 60, 62, 65, 67, 70, 75, 85, 90, 91, 92, etc. (în sensul, cîte se vor mai construi). În cadrul unei serii există posibilitatea extinderii sistemului de calcul, fie prin înlocuirea unităţii centrale, fie prin adăugarea de echipamente periferice noi, fie prin amîndouă. Aceasta înseamnă că sistemul de calcul are *natură modulară*.

1.2. PĂRŢILE COMPONENTE ALE UNUI SISTEM DE CALCUL. PURTĂTORI DE INFORMAŢIE

1.2.1. Părţile componente ale unui sistem de calcul

Pe parcursul întregii cărţi ne referim numai la sistemele electronice de calcul, înzestrate cu un calculator numeric universal. Calculatorul este un mijloc de calcul automat care permite să se efec-

tueze un şir finit de operaţii, conform unui program de lucru determinat. Programul este constituit din instrucţiuni şi realizează prelucrarea informaţiei conform dorinţei utilizatorului şi a posibilităţilor sistemului de calcul. Pentru prelucrarea informaţiei sînt necesare următoarele trei părţi ale sistemului: dispozitivul de introducere sau de intrare, dispozitivul de prelucrare şi dispozitivul de ieşire sau de extragere a rezultatelor.

Dispozitivul de prelucrare formează *unitatea centrală* a sistemului. Ea se compune din:

— memoria principală;
— unitatea de comandă a unităţii centrale;
— unitatea aritmetică şi logică.

Legăturile informaţionale dintre dispozitivele menţionate se văd în figura 1.

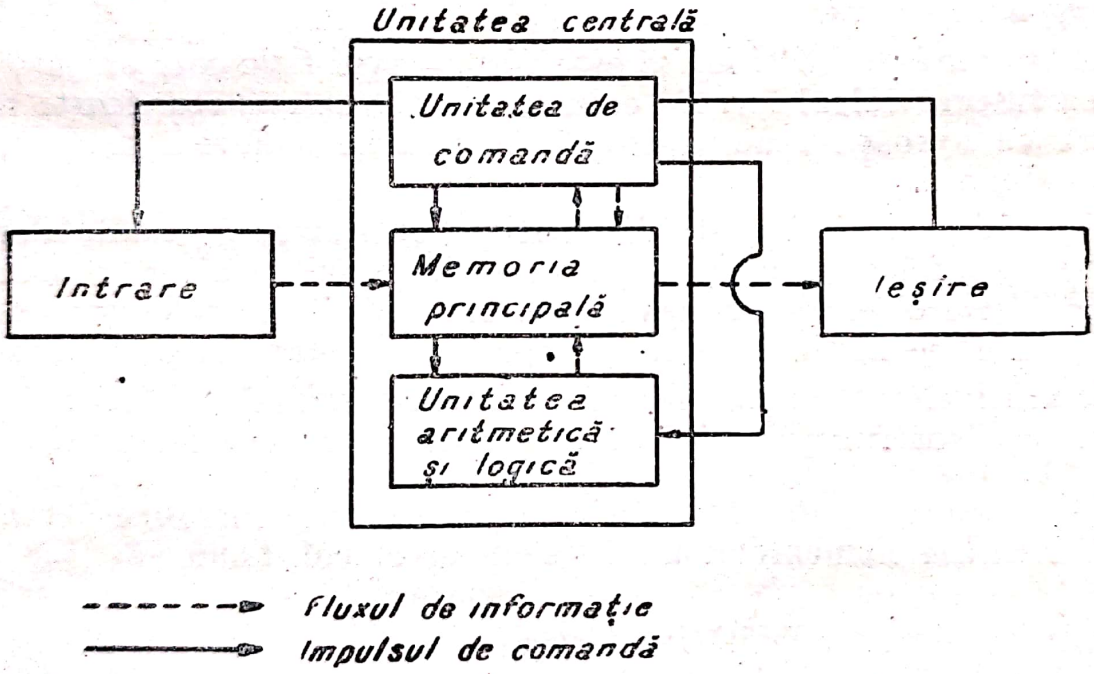


Fig. 1. Părţile componente ale unui sistem de calcul

1.2.2. Descrierea părţilor componente

Dispozitivul de intrare asigură introducerea programului şi a datelor de prelucrat în memoria principală. Există mai multe tipuri, funcţie de mediul-suport purtător al informaţiei, pe care aceasta este înregistrată. Cele mai obişnuite sînt cititoarele de cartele perforate, cititoarele de bandă perforată sau cititoarele de bandă magnetică.

Scanned with OKEN Scanner

Memoria principală sau memoria internă, sau pe scurt memoria, constituie locul unde se depozitează toate elementele de informaţie: programul şi datele programului, precum şi rezultatele intermediare. Memoria este constituită dintr-un anumit număr de locuri (locaţii) de memorie, numite şi celule de memorie, numerotate în ordine crescătoare. Deci, fiecărui loc de memorie i se asociază o *adresă*. Cunoaşterea unei *adrese a* permite citirea conţinutului din locul cu această adresă sau înscrierea unei informaţii în acest loc. Prin citire, conţinutul celulei nu se distruge.

Unitatea de comandă a unităţii centrale descifrează într-o anumită ordine instrucţiunile programului memorat şi indică unităţii aritmetice şi logice ce operaţie trebuie să execute şi, de asemenea, ce au de făcut dispozitivele de intrare/ieşire. Potrivit acestor funcţiuni, unitatea de comandă este formată din *decodificatorul de instrucţiuni* şi din *generatorul de comenzi*.

Unitatea aritmetică şi logică este dispozitivul fizic care asigură efectuarea tuturor calculelor. Ea este, în esenţă, un sumator foarte rapid, pentru că aproape toate operaţiile aritmetice şi logice se realizează prin adunare.

Dispozitivul de ieşire este dispozitivul de livrare a rezultatelor. La ieşire se pot folosi ca medii-suport cartele, benzi de hîrtie, benzi magnetice, hîrtie obişnuită de scris, etc. Cel mai des se foloseşte livrarea rezultatelor pe hîrtie de scris şi tipărirea acestora la o imprimantă.

Dispozitivele de intrare-ieşire formează *perifericele* sistemului de calcul, a căror funcţionare este asigurată de unitatea centrală.

1.2.3. Ierarhia elementelor unui sistem de calcul. Canale

Viteza de lucru a unităţii centrale este foarte mare în comparaţie cu viteza de lucru a elementelor periferice. Comandarea de unitatea centrală a *tuturor* acţiunilor perifericelor înseamnă folosirea neraţională a timpului de calcul. De aceea, la calculatoarele actuale moderne, o mare parte din sarcinile unităţii centrale de comandare a perifericelor a fost preluată de *canale*. Canelele sînt legate de periferice prin intermediul *unităţilor de comandă ale perifericelor*. Canalul este dispozitivul fizic care asigură comanda transferului de informaţie de la unitatea centrală la unitatea de comandă a perifericului şi invers. Unitatea de comandă a perifericului asigură comanda transportului de informaţie de la canal la periferic şi invers. Deci ierarhia elementelor unui sistem de calcul se prezintă astfel:

Scanned with OKEN Scanner

— la o unitate centrală se pot conecta mai multe canale ;

— la un canal se pot conecta mai multe unităţi de comandă ale perifericelor ;

— la o unitate de comandă a perifericelor se pot conecta mai multe periferice.

În continuare vom numi periferice toate dispozitivele sistemului, care diferă de unitatea centrală, de canale şi de unităţile de comandă ale perifericelor. Pentru fiecare sistem de calcul există un număr maxim şi un număr minim de periferice, de unităţi de comandă a perifericelor şi de canale, care pot fi conectate la o unitate centrală. În această accepţie, vom numi *configuraţie de calcul* totalitatea dispozitivelor de intrare, prelucrare şi ieşire care se găsesc montate în sala unui calculator;

Tehnicile de comandare a unităţilor de intrare-ieşire sînt în continuă perfecţionare. Totuşi se pot distinge două clase mari de posibilităţi :

1. Unitatea de comandă a unităţii centrale este comună pentru programul care trebuie executat şi pentru unităţile de intrare-ieşire. Comanda este comutată în mod continuu de la program la aceste unităţi şi invers, ceea ce duce la un consum mare de timp.

2. Se folosesc canale pentru comandarea perifericelor, care preiau o parte din sarcinile unităţii centrale.

Relativ la autonomia în funcţionare a unui canal, există două tipuri de canale :

a) *Canale cu funcţiuni simple, independente.* Unitatea de comandă a unităţii centrale pune la dispoziţia canalului *toate* informaţiile necesare executării operaţiilor de intrare/ieşire : operaţia care trebuie executată-citire sau scriere, lungimea cîmpului de date care trebuie manevrat, de unde sînt luate datele, unde sînt depuse rezultatele. Canalul execută toate aceste operaţii, în *paralel* cu desfăşurarea programului în unitatea centrală. Acest mod de lucru prezintă dezavantajul că unitatea centrală este ocupată *destul de mult* cu urmărirea operaţiilor de intrare-ieşire.

b) *Canalele cu funcţiuni complexe, independente.* Canalul îşi desfăşoară activitatea pe baza unui *program propriu,* care este depus în memoria principală. Acest program se actualizează de fiecare dată pe baza unor comenzi sumare primite de la unitatea centrală. Aceste comenzi constau în următoarele informaţii : trebuie executată o operaţie de intrare sau de ieşire, care anume unitate periferică este solicitată, la ce adresă se găseşte programul de canal în memorie. Canalul execută toate operaţiile de intrare-ieşire *independent* şi *în paralel* cu programul care este executat de unitatea centrală. Programul de canal este format

Scanned with OKEN Scanner

din instrucțiuni, numite *cuvinte de canal*. Deci canalul este un mic calculator, care execută toate operațiile de intrare-ieșire, eliberînd de această sarcină unitatea centrală.

La calculatorul Bull Gamma 60, unitatea în funcționarea sistemului este asigurată prin conectarea tuturor elementelor cu memoria centrală, la care are acces fiecare element în ambele sensuri, prin două canale numite *canalul colector* și *canalul distribuitor*. Acestea sînt comandate de distribuitorul de program prin intermediul unui program local ([10], cap. II) .

Sistemul IBM 7094 posedă canale cu funcțiuni complexe, care, datorită funcționării lor automate, permit să se efectueze simultan operații de intrare, de ieșire și de calcul. La o unitate centrală se pot conecta cel mult opt canale. Fiecare canal este adresabil separat, ca și unitatea periferică de care este legat.

La sistemul IBM 360 ierarhia elementelor discutate se prezintă astfel:

— numărul canalelor conectabile este cel mult 7, numerotate 0—6;

— numărul unităților de comandă ale perifericelor, conectabile la un canal, este cel mult 16, numerotate 0—15;

— numărul perifericelor conectabile la o unitate de comandă a acestora este cel mult 16, numerotate 0—15.

Deci, într-o configurație concretă de calcul, toate elementele au adrese determinate de numerele pe care le au în ierarhia prezentată.

Sistemul IBM 360 este de *natură modulară*, adică o configurație dată se poate modifica prin conectarea sau deconectarea unor canale, unități de comandă a perifericelor și a perifericelor.

1.2.4. Tipuri de canale

După modul de funcționare și viteza de transfer a informației, există două tipuri de canale: *canalul selector* și *canal multiplexor*.

Un canal selector deservește într-un moment dat o unitate de comandă a perifericelor, iar aceasta deservește un singur periferic. Canalul selector este un canal rapid, care lucrează cu periferice rapide, ca discurile magnetice și benzile magnetice. Transmiterea informației se face pe seturi, într-un timp foarte scurt. Unitățile de comandă care deservesc un canal selector sînt în general unități specializate, de exemplu, numai pentru discuri sau numai pentru benzi.

Un canal multiplexor are mai multe *subcanale*. El deservește într-un moment dat mai multe unități de comandă ale perifericelor, iar o unitate de comandă deservește într-un moment dat un singur periferic. Transmiterea informației se face pe unități de informații, de exemplu

Scanned with OKEN Scanner

bait cu bait, unde 1 bait = 8 biți. Canalele multiplexoare sînt conectate cu perificele lente : cititor de cartele, perforator de cartele, consola unității centrale*, imprimanta pentru obținerea rezultatelor tipărite.

1.2.5. Purtători de informație

Există multe tipuri de purtători de informație, adică de medii-suport pe care se înregistrează informația inițială care se introduce în unitatea centrală sau se extrage din unitatea centrală.

Cartela este cel mai obișnuit purtător de informație. Există două tipuri de cartele mai des folosite: cele lansate pentru echipamentul IBM și cele lansate pentru echipamentul Remington-Rand. Între aceste două tipuri există mici diferențe specifice, principiile de bază fiind aceleași.

Prezentăm numai cartela de tip IBM. Aceasta are o formă dreptunghiulară cu lungimea împărțită în 80 de coloane, numerotate 1—80, iar lățimea împărțită în 12 rînduri, numerotate 12, 11, 0, 1, 2, ..., 9. Pentru memorarea pe cartelă a unui caracter, se folosește o coloană. Deci pe o cartelă pot fi memorate 80 de caractere. Asocierea biunivocă a fiecărui caracter cu o anumită combinație de perforații pe coloană se numește *codificare.*

Rîndurile 12, 11, 0 formează partea de zonă a cartelei, iar 1, 2, ..., 9 formează partea de cifră. Menționăm că, la codificarea cifrelor zecimale 0, 1, ..., 9, rîndul 0 este tot parte de cifră.

Există numeroase coduri de codificare a caracterelor. Indicăm codul folosit de sistemul IBM 360. Codificarea literelor mari ale alfabetului englez $A—Z$ și a cifrelor zecimale 0—9 se face conform tabelului 1. De exemplu, litera B va fi înregistrată pe cartelă în următorul mod: va exista o perforație pe rîndul 12 și o perforație pe rîndul 2; R va avea o perforație pe rîndul 11 și una pe rîndul 9.

TABELUL 1

Grupa	Combinații de perforații	
	zona	cifra
$A - I$	12	1—9
$J - R$	11	1—9
$S - Z$	0	2—9
0 — 9	nimic	0—9

*) Consola este un periferic de comunicare în ambele sensuri între unitatea centrală și utilizator.

Scanned with OKEN Scanner

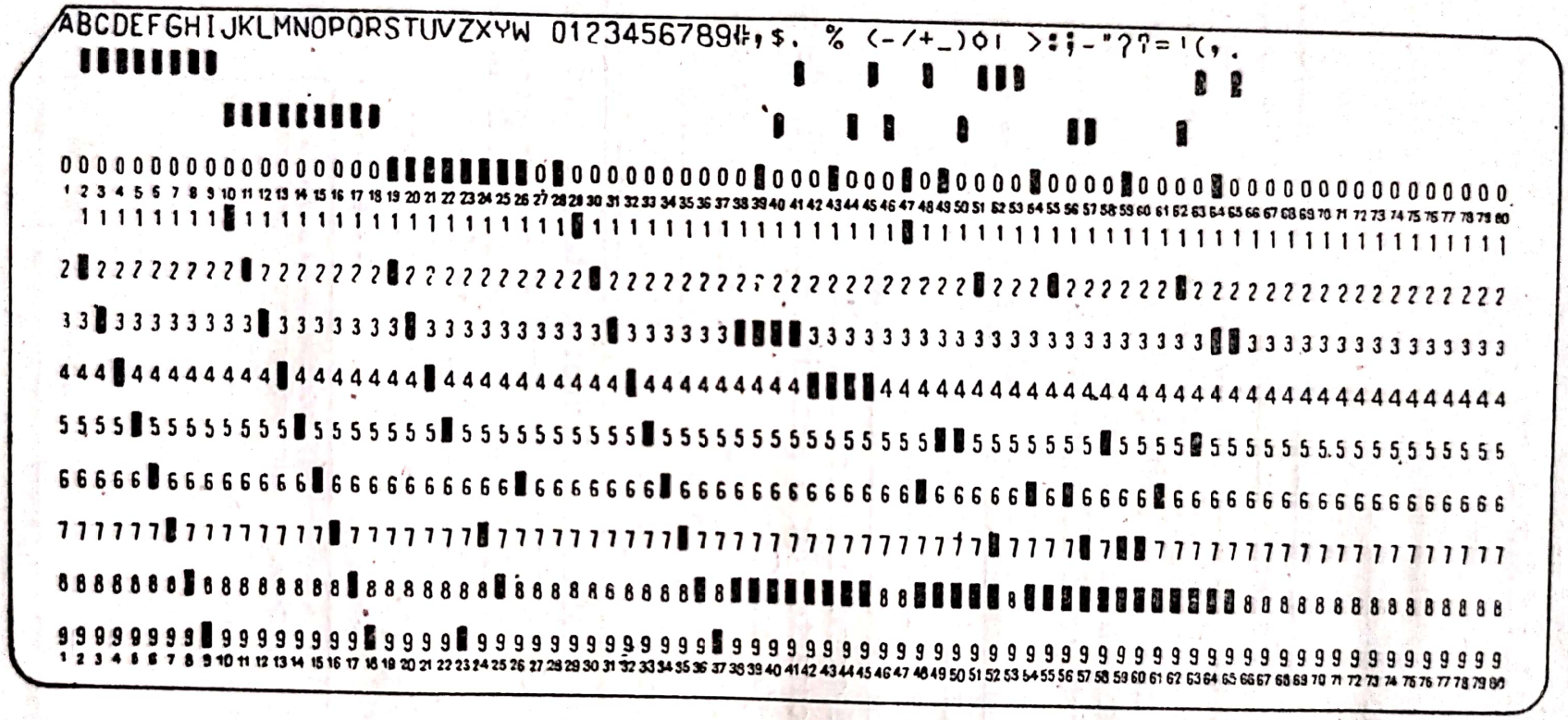


Fig. 2. Codificarea și înregistrarea caracterelor pe cartelă

Figura 2 arată codificarea și înregistrarea caracterelor pe cartelă. Numărul combinațiilor posibile de perforații în partea de zonă și/sau partea de cifră este mult mai mare decît numărul total de caractere pe care îl folosește un sistem de calcul.

Pentru trecerea informației — programul sau datele — pe cartele, se folosește perforatorul cu taste, acționate manual. Spre a se evita trimiterea informațiilor la perforator pentru a fi trecute pe cartele, a apărut *metoda perforării manuale* pe *cartele semiperforate* care prezintă pe linii și pe coloane înțepături fine ce determină o rețea de dreptunghiuri. Cartela semiperforată se plasează pe un suport IBM special și se realizează perforarea prin apăsarea manuală cu un creion. Pentru o mai mare siguranță, cartela semiperforată are numai 40 de coloane. Codul de perforare este același ca la cartelele cu 80 de coloane.

Metoda perforării manuale cîștigă tot mai mult teren datorită economiei mari de timp pe care o introduce.

Banda de hîrtie perforată este un alt mediu-suport de informație. Există două tipuri fundamentale de bandă de hîrtie: bandă cu opt piste și bandă cu cinci piste.

La ambele tipuri, unul din canale este canalul de *control al parității*. În acest canal va exista o perforație, numai dacă caracterul care trebuie memorat necesită un număr impar de perforații. Deci totdeauna un caracter memorat pe banda de hîrtie va avea asociat un număr par de perforații în canale benzi. Înregistrarea caracterelor pe bandă se face după un cod bine determinat. Banda de hîrtie este un mediu continuu de înregistrare, pe care se pot înregistra seturi de informație, teoretic, de orice mărimi. Ea prezintă însă dezavantajul că manevrarea ei este mai puțin flexibilă decît a cartelelor. Acest lucru se vede în primul rînd la corectarea informației înregistrate.

Rapoartele imprimate sînt cel mai obișnuit mijloc de obținere a rezultatelor. Tipărirea se face de o imprimantă specială, conform dorințelor utilizatorului, pe care acesta o livrează sistemului de calcul prin programul său.

Banda magnetică este un purtător de informație foarte folosit de sistemele de calcul, mijlocii și mari. Ea se aseamănă cu banda de magnetofon, dar are alte dimensiuni. Există benzi cu șapte piste și cu nouă piste. Una din piste este pista de paritate, care face ca fiecărui caracter înregistrat pe bandă să îi fie asociat un număr par de locuri magnetizate. Controlul parității este asigurat de unitatea de comandă a benzii magnetice.

Informația se memorează pe bandă sub formă de blocuri de lungime fixă sau variabilă. Blocurile sînt despărțite prin spații goale. Spațiile sînt necesare pentru pornirea și/sau oprirea benzii, la citire și/sau scriere. Pentru trecerea uniformă a benzii prin fața capului de citire/

Scanned with OKEN Scanner

scriere, banda trece prin două canale cu vacuum. Prezentarea schematică a unei unităţi de bandă magnetică se vede în figura 3.

Există benzi de diferite dimensiuni. Cele mai folosite sînt rolele de bandă cu lungimea de 732 m şi cu lăţimea benzii de 1,5 cm. Densitatea informaţiei pe centimetru nu este unică la toate tipurile de bandă.

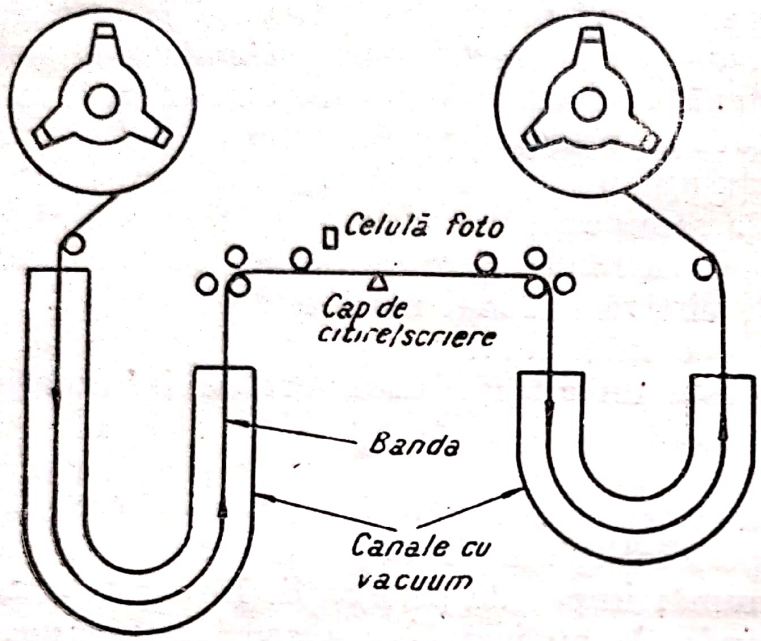


Fig. 3. Unitate de bandă magnetică

Funcţie de sistemul utilizat, un ţol de bandă (2,54 cm) poate conţine de la 200 la 2000 caractere. Densităţile cele mai folosite sînt 320 caractere pe cm sau 640 caractere pe cm. O rolă de lungimea de 732 m poate conţine pînă la 14 milioane de caractere. Ea poate fi citită sau scrisă în mai puţin de cinci minute.

Unităţile de bandă magnetică intră în compunerea memoriei externe a unui sistem de calcul.

Discurile magnetice. Acestea sînt grupate în pachete de discuri magnetice, într-un număr care depinde de tipul unităţii periferice de discuri. Discurile intră în constituirea memoriei externe a unui sistem de calcul.

Un pachet de discuri este format dintr-un număr de discuri, dispuse pe un ax vertical, la o distanţă de cîţiva milimetri unul de altul. Fiecare faţă a unui disc are un număr de cercuri concentrice numite piste, pe care se înregistrează informaţia. Pentru mai multă claritate, ne vom referi în mod concret la pachetele de discuri tip 2311 foarte folosite de sistemul IBM 360.

Fiecare pachet de discuri conţine şase discuri, deci în total 12 feţe. Faţa 1 şi faţa 12 sînt feţe de protecţie pe care nu se înregistrează informaţia. Pe fiecare faţă există 203 piste, numerotate 0—202. Totalitatea pistelor cu aceeaşi rază, de pe cele 10 feţe, formează un *cilindru.* Cei 203 cilindri sînt numerotaţi 0—202. Sînt utilizabili cilindrii 0—199, iar ceilalţi trei cilindri sînt de rezervă. Atunci cînd se defectează o pistă de pe cilindrii utilizabili, ea este înlocuită *automat,* prin intermediul unităţii centrale şi a unităţii de comandă a perifericului, cu o pistă de rezervă.

Scanned with OKEN Scanner

Capacitatea de memorare a unei piste nu depinde de raza ei. Pistele cu raze diferite au densităţi de memorare diferite. Un pachet 2311 poate memora pînă la 7,25 milioane baiţi.

Funcţie de tipul unităţii de discuri folosite, pachetele pot fi interschimbabile sau neinterschimbabile. Ultimele sînt montate definitiv într-o unitate de discuri şi fiecare pachet are mai mult de şase discuri. La unităţile cu discuri interschimbabile, un pachet de discuri poate fi montat (demontat) pe (de pe) orice unitate periferică, legată la unitatea de comandă a discurilor.

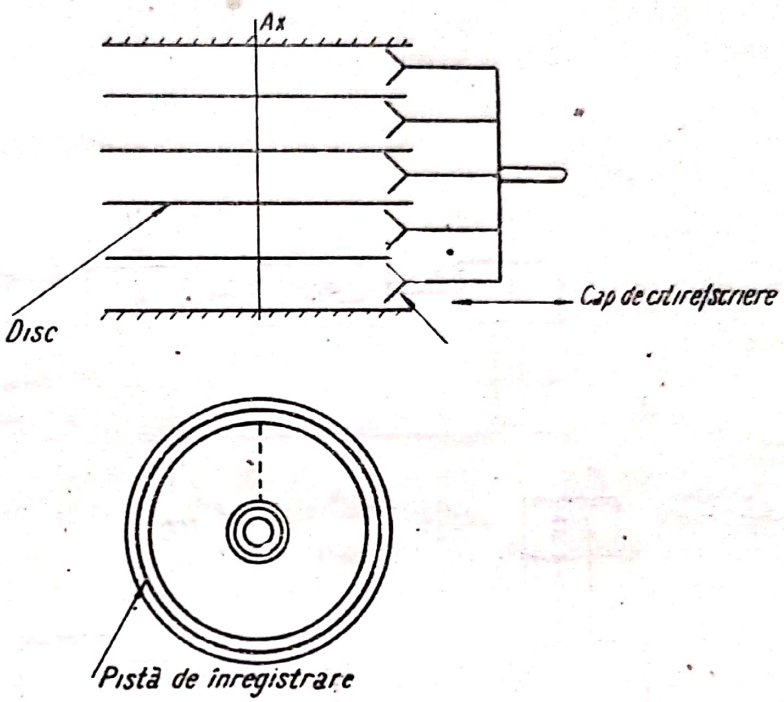


Fig. 4. Schema unui pachet de discuri

Discurile magnetice sînt memorii cu acces rapid la informaţia înregistrată. Citirea şi scrierea se face cu ajutorul unor capete de citire — scriere, atîtea cîte feţe utilizabile există şi sînt montate pe un braţ care se deplasează în ambele sensuri în plan orizontal. Pachetul de discuri se roteşte cu cîteva mii de rotaţii pe minut, asigurînd trecerea informaţiei prin faţa capului de citire-scriere.

Figura 4 prezintă schematic un pachet de discuri.

Pentru că la memoriile pe discuri există posibilitatea accesului la informaţie în orice loc de pe disc şi în orice moment, ele se numesc *memorii externe cu acces direct*. Tot memorii externe cu acces direct mai sînt memoriile pe *folii magnetice* şi pe *tambur magnetic* [10].

1.2.6. Configuraţia de calcul de la Centrul de Calcul al Universităţii din Bucureşti

Acest centru de calcul este dotat cu un sistem de calcul IBM 360, model 30. Configuraţia prezentă de calcul se vede în figura 5.*

Am notat prin *C* — canalul şi prin *UC* — unitatea de comandă a perifericului.

*) Este vorba de configuraţia existentă la data de 1 februarie 1972.

Scanned with OKEN Scanner

Din figură se desprind următoarele:

— există două canale: canalul 0 — multiplexor, 1 — selector;

— există două unități de benzi magnetice, tipul 2400 și patru unități de discuri magnetice, tipul 2311;

— adresele perifericelor sînt: cititorul de cartele — *00C*, perfora-

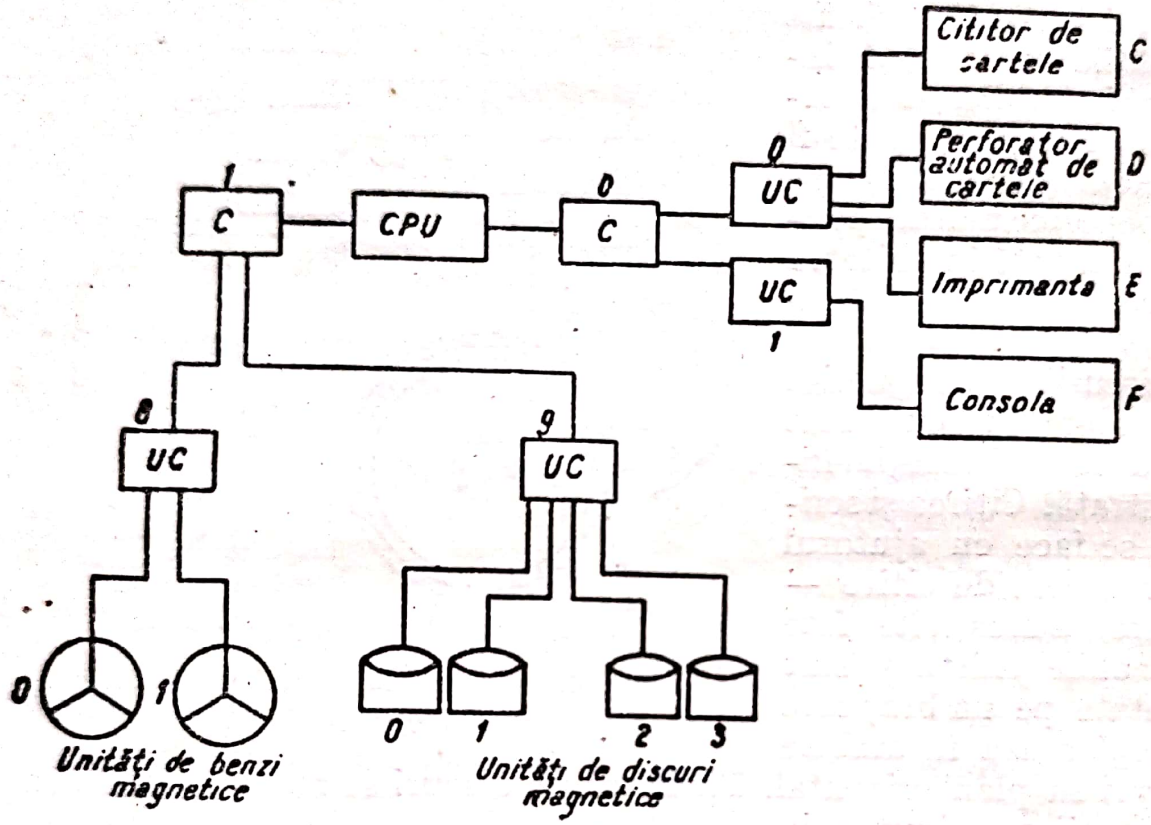


Fig. 5. Configurația de la Centrul de calcul al Universității din București

torul de cartele — *00D*, imprimanta — *00E*, consola unității centrale — *01F*, unitățile de benzi — 180, 181, unitățile de discuri magnetice 190, 191, 192, 193.

Perifericele legate de canalul 0 sînt periferice lente. Cititorul, perforatorul și imprimanta au, fiecare, memorie proprie, numită *memorie tampon*, necesară activității canalului.

Cititorul de cartele are tipul 2540 și constituie dispozitivul de intrare pentru program și pentru date. Viteza de citire este de 1000 cartele/minut, adică 1330 baiți/s. Perforatorul de cartele are tipul 2540 și poate fi folosit ca periferic de ieșire. El este un perforator automat, comandat prin program de utilizator, spre deosebire de perforatorul

Scanned with OKEN Scanner

manual, folosit la trecerea programului pe cartele. Imprimanta are tipul 1403 şi constituie perifericul de ieşire atunci cînd se doreşte tipărirea rezultatelor pe coli de hîrtie (rapoarte imprimate). Imprimarea se face cîte un rînd întreg la un moment dat. Viteza de imprimare este de 1100 rînduri/minut. Unităţile de benzi folosesc benzi cu nouă piste. Lungimea spaţiilor goale dintre blocurile de informaţie este de 1,5 cm. Pachetele de discuri pentru unităţile 2311 sînt interschimbabile.

1.3. BAZA ARITMETICĂ A UNUI CALCULATOR

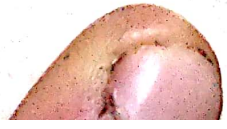
1.3.1. Sisteme de numeraţie. Convertirea numerelor

Dacă baza de numeraţie este b, pentru scrierea oricărui număr, se folosesc cifrele 0, 1, 2, ..., $b - 1$. De exemplu, *sistemul zecimal* are $b = 10$ şi foloseşte cifrele zecimale 0,1, ..., 9. Numărul $409 = 4 \times 10^2 + 0 \times 10^1 + 9 \times 10^0$.

Calculatoarele electronice folosesc dispozitive speciale de memorare a cifrelor oricărui număr. Aceste dispozitive au numai două stări posibile: un tub electronic conduce sau nu conduce, un circuit este închis sau deschis, un element cu proprietăţi magnetice este magnetizat sau nemagnetizat. Stările sînt stabile şi deci se impune folosirea bazei $b = 2$, adică a *sistemului binar* cu cifrele 0 şi 1, numite *biţi* de informaţie. De exemplu, numărul binar 10111 este format din cinci biţi, aşa cum numărul 31094 este format din cinci cifre zecimale.

Convertirea unui număr N din sistemul zecimal în sistemul binar se face prin împărţiri succesive cu 2, şi anume: se împarte N la 2, apoi cîtul obţinut din nou se împarte la 2 ş.a.m.d. Calculele se fac în baza 10. Prin scrierea resturilor împărţirilor de la stînga la dreapta, începînd cu ultimul rest, se obţine N convertit în binar. $N = 467$ se reprezintă ca 111010011 în binar. Pentru a face proba, se foloseşte scrierea în baza 2: $111010011 = 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 256 + 128 + 64 + + 16 + 2 + 1 = 467$. Proba a arătat, de fapt, cum se face trecerea de la sistemul binar la sistemul zecimal.

Întocmai ca şi în sistemul zecimal, în sistemul binar se întocmesc tabele de adunare şi de înmulţire ale cifrelor binare. Acestea se văd în tabelul 2, unde prin * am notat faptul că apare o cifră de transport.



TABELUL 2

Adunarea şi înmulţirea cifrelor binare

+	0	1
0	0	1
1	1	0*

×	0	1
0	0	0
1	0	1

Pentru numerotarea adreselor memoriei principale, se foloseşte *sistemul hexazecimal*, adică sistemul cu baza $b = 16$. Cifrele posibile sînt $0, 1, \ldots, 15$. Spre a nu se confunda cifrele hexazecimale cu cifrele zecimale, le vom scrie astfel: $0 = 0$, $1 = 1$, $2 = 2$, $\ldots$, $9 = 9$, $10 = A$, $11 = B$, $12 = C$, $13 = D$, $14 = E$, $15 = F$. Convertirea din zecimal în hexazecimal se face prin împărţiri succesive cu 16, cu calculele în baza 10. De exemplu, numărul zecimal $N = 492$ se scrie în hexazecimal ca $1(14)(12)$ sau $1EC$. Pentru probă, adică pentru convertirea din hexazecimal în zecimal, se foloseşte scrierea în baza 16 cu calculele în baza 10, adică $1EC = 1 \times 16^2 + E \times 16^1 + C \times 16^0 = 1 \times 256 + 14 \times 16 + 12 = 492$. Tabelele de adunare şi de înmulţire a cifrelor hexazecimale se întocmesc uşor.

În general, putem spune că, pentru a converti numerele din baza veche b în baza nouă h, există două metode:

1. trecerea de la baza b la baza h cu calculele în b;
2. trecerea de la baza b la baza h cu calculele în h.

S-ar părea că mai există o metodă de trecere de la baza b la h, prin efectuarea calculelor într-o bază g, $g \neq b$, $g \neq h$. Această metodă rezultă din 1 şi 2 în felul următor: se trece numărul din baza b în baza g, cu calculele în g (metoda 2), apoi numărul obţinut în baza g se trece în baza h, cu efectuarea calculelor în baza g (metoda 1).

Vom ilustra metodele 1 şi 2 prin exemple.

Metoda 1: Să se convertească numărul $N = 13201301$, scris în baza $b = 4$, în baza $h = 5$, cu calculele în baza 4. Paşii de calcul sînt următorii: a) Scriem pe $h = 5$ în baza $b = 4$ şi obţinem 11. b) Facem împărţirile succesive sau, altfel spus, scriem algoritmul sistemelor de numeraţie pentru N şi h scrise în baza 4. Calculele făcute în baza 4 arată astfel:

```
13201301    11
11          1200112    11
=22         11         103101    11
 22         =100        33       3312    11
 ==013        33       =101      33      301    11
    11       ==11        33      ==12     22     21    11
  ==20         11       ==20       11     =21    11    1
    11        ==12        11       =1      11    10
   =31          11       =31              10
    22         =1         22
   =3                    =3
```

Începînd cu ultimul rest se obţine numărul $N = 1(10)(10)1313$. c) Transformăm fiecare cifră a acestui număr, scris în baza $b = 4$, în baza $h = 5$. Atunci, cifrele (10) devin 4. Deci se obţine numărul $N = 1441313$ în baza 5.

Propunem cititorului să facă proba prin intermediul bazei 10 şi, direct, prin convertirea în baza 4 a numărului obţinut în baza 5.

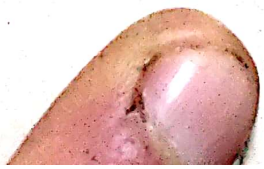
Metoda 2: Să se convertească $N = 21101$, scris în baza $b = 3$, în baza $h = 2$, cu calculele în baza 2. Paşii de calcul sînt următorii: a) Scriem $b = 3$ din baza 3 în baza $h = 2$ şi obţinem $3_3 = 11_2$. b) Convertim fiecare cifră a lui N din baza 3 în baza 2. Atunci 2, 1, 1, 0, 1 devin 10, 1, 1, 0, 1. c) Scriem algoritmul de calcul al lui N, adică $N = 10 \times 11^4 + 1 \times 11^3 + 1 \times 11^2 + 0 \times 11^1 + 1 \times 11^0$. Calculele se fac în baza 2 şi se obţine $N = 11000111$ scris în baza 2. Proba rămîne din nou în seama cititorului.

Pînă acum ne-am ocupat numai de convertirea numerelor întregi. Convertirea numerelor fracţionare subunitare, în valoare absolută, se face într-un mod aproape analog cu convertirea numerelor întregi, şi anume, prin înmulţiri succesive cu baza în care dorim să apară numărul scris. Se vor considera cifre ale numărului în baza nouă h cifrele în baza h care depăşesc spre stînga, la înmulţire, poziţia virgulei din numărul în baza veche b. Dacă convertirea se face exact, de la un anumit pas se obţine mereu cifra zero ca cifră a numărului în baza h. Vom da cîteva exemple.

Exemplul 1. Convertirea în binar a numărului fracţionar zecimal 0,35 se face după schema:

```
0,35
 X2
------
0 70
 X2
------
1 40
 X2
------
0 80
 X2
------
1 60
 X2
------
1 20
```

Procesul poate continua. S-a obţinut ca $0{,}35_{10} \approx 0{,}01011$. Proba, cu calculele în baza 10, se face astfel: $0{,}01011 = 0/2^1 + 1/2^2 + 0/2^3 + 1/2^4 + 1/2^5 = 0{,}343$.



Scanned with OKEN Scanner

Exemplul 2. Să se convertească 0,13 în hexazecimal. Se fac calculele în zecimal, ca în schema următoare:

```
 0,13
X 16
----
  78
 1 3
----
 2 08
X 16
----
  48
   8
----
 1 28
X 16
----
 1 68
 2 8
----
 4 48
X 16
----
 2 88
 4 8
----
 7 68
```

Calculele pot continua. Pînă la acest pas s-a obţinut ca $0{,}13_{10} \approx 0{,}2147_{16}$. Se observă că, de fiecare dată, se înmulţeşte numai partea zecimală, din dreapta poziţiei virgulei.

Se poate face proba prin efectuarea calculelor în baza 10.

Pentru alte aspecte ale problemelor de convertire, se poate consulta [7].

Atragem atenţia că datele numerice pe care utilizatorul le transmite sistemului de calcul sînt în sistemul zecimal. Convertirea în binar se face automat, prin intermediul canalului şi a unităţii centrale. Pentru a fi convertite în binar şi apoi depuse în memorie principală, datele sînt luate din memoria tampon.

Pentru o convertire rapidă din zecimal în binar se foloseşte baza intermediară 16, în felul următor: se transformă numărul zecimal în hexazecimal, apoi fiecare cifră hexazecimală obţinută se transformă în binar cu ajutorul a patru biţi. Numărul obţinut este tocmai reprezentarea în binar a numărului zecimal iniţial.

De exemplu, $99_{10} = 63_{16} = 0100011_2$, pentru că 6 se scrie cu patru biţi, 0110, iar 3 se scrie 0011.

De multe ori se pune problema de a şti cîte cifre generează, în baza h, un număr pentru care se cunoaşte numărul de cifre în baza b. Se demonstrează că dacă în scrierea în baza b a unui număr natural N, prima cifră din stînga fiind nenulă, se află m cifre, atunci în scrierea

în baza h a aceluiași număr N, cu prima cifră din stînga tot nenulă, se află l cifre, unde l verifică relația:

$\left[\frac{m-1}{\log_b h}\right] + 1 \leqslant l \leqslant -\left[-\frac{m}{\log_b h}\right]$, unde prin [ ] am notat funcția, parte întreagă. Această observație este foarte importantă atunci cînd se stabilește lungimea unui registru, cunoscîndu-se ordinul de mărime al numerelor cu care se va lucra.

1.3.2. Tipuri de prelucrare și coduri de reprezentare a datelor numerice

Pentru memorarea unui număr se folosește o mulțime ordonată și finită de dispozitive de memorie, numită *registru de memorie* sau *registru*. Unul din dispozitive — de obicei, primul din registre — memorează semnul, iar fiecare din celelalte memorează numai o cifră de un anumit rang în scrierea în baza 2. Pentru semnele + și — se stabilește corespondența cu 0 și 1.

Prelucrarea numerelor întregi este relativ simplă. Dacă se presupune că cel mai mare număr întreg folosit în calcule este N, atunci se construiesc registre care pot memora toate cifrele lui N. De exemplu, dacă registrul are 32 de biți, primul bit fiind bitul semn, cel mai mare număr care se poate înscrie în 31 de biți este $N = 2^{31} - 1 = 2\,147\,483\,647$. La un astfel de calculator, se anulează orice program care generează numere întregi mai mari decît numărul indicat mai sus, aceasta însemnînd depășirea capacității registrului.

Prelucrarea numerelor reale impune folosirea virgulei zecimale, a cărei poziție se poate schimba o dată cu schimbarea numărului real prelucrat. Ținerea automată a evidenței acestei schimbări în dispozitivul aritmetic nu este un lucru ușor. De aceea, pentru a ști unde este plasată virgula, s-a făcut o convenție, care a condus la două tipuri de prelucrare: *prelucrarea în virgulă fixă* și *prelucrarea în virgulă mobilă*.

La prelucrarea în virgulă fixă, virgula este plasată imediat după bitul semn, adică se folosesc numai numere subunitare. Această restricție nu restrînge în nici un fel domeniul de aplicabilitate al prelucrării, pentru că dacă în calcule intervin numere reale cu un număr oarecare de cifre înainte de virgulă, aceste numere se împart în prealabil cu un număr convenabil ales. După natura calculelor, rezultatul trebuie apoi corectat prin înmulțire cu un anumit număr. De asemenea, trebuie să se cerceteze și rezultatele parțiale ale calculelor, pentru a se vedea dacă va avea loc sau nu va avea loc o depășire a unității. Dacă depășirea va avea loc, atunci rezultatul parțial respectiv va fi

Scanned with OKEN Scanner

împărţit cu un număr convenabil ales şi se va ţine cont de aceasta la corecţia finală.

Prelucrarea în virgulă fixă este deci incomodă. Totuşi, se foloseşte în calculele în care ordinele de mărime ale numerelor de prelucrat sînt cunoscute.

Prelucrarea în virgulă mobilă se foloseşte atunci cînd ordinele de mărime sînt variabile şi necunoscute integral apriori. Numerele sînt reprezentate sub formă semi-logaritmică. Astfel, numărul real a se scrie $a = b^c \cdot m$, unde b este baza sistemului de numeraţie, iar c şi m sînt, respectiv, *caracteristica* şi *mantisa* lui a. Caracteristica c este un întreg pozitiv, zero sau negativ, iar m este un număr subunitar. m se memorează ca în virgulă fixă.

Pentru a se efectua o adunare (scădere) este necesar ca termenii să fie aduşi la aceeaşi caracteristică. Astfel, în sistemul zecimal, $a = 18{,}92$ şi $d = 394{,}72$ conduc la: $a + d = 10^2 \times 0{,}1892 + 10^3 \times 0{,}39472 = 10^3 \times 0{,}01892 + 10^3 \times 0{,}39472 = 10^3 \times (0{,}01892 + 0{,}39472)$. La înmulţire (împărţire), aducerea la aceeaşi caracteristică nu este necesară.

Pentru mărirea preciziei calculelor, numerele în virgulă mobilă se reprezintă sub formă *normalizată*, adică caracteristica este astfel aleasă încît mantisa să aibă prima cifră după virgulă nenulă. Aceasta se exprimă astfel: $\frac{1}{b} \leqslant |m| < 1$. În calculatoarele actuale, reprezentarea numerelor reale cu ajutorul caracteristicii şi a mantisei, sub formă normalizată, se face automat fără intervenţia programului.

Memorarea numerelor în virgulă mobilă se face în registre de virgulă mobilă. Aceste registre sînt împărţite în mai multe cîmpuri de biţi, şi anume: primul bit din stînga memorează semnul numărului real a; urmează un număr de biţi care memorează cifrele lui c, împreună cu semnul lui c; ultima parte memorează cifrele lui m. Lungimea registrului de virgulă mobilă şi împărţirea sa în cîmpuri este funcţie de magnitudinea numerelor de prelucrat şi de precizia (numărul de cifre după virgulă) dorită.

Fie, de exemplu, un calculator cu registrul de 45 biţi distribuiţi ca în figura 6.

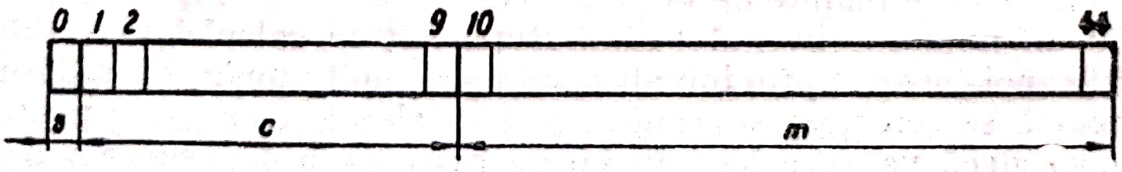


Fig. 6.

Scanned with OKEN Scanner

Biții 0 și 1 memorează semnul lui a, respectiv semnul caracteristicii. Cum, în biții 2—9, numărul maxim care poate fi înscris este $2^8 - 1 = 255$, acest calculator poate să prelucreze numere ale căror valori absolute pot să fie de maximum $(2^{35}-1)\times 2^{255}$ sau de minimum 2^{-255}. Un asemenea interval se consideră că acoperă aproape toate numerele care trebuie prelucrate în problemele concrete, fără nici o precauție specială din partea programatorului.

Există calculatoare ale căror registre de virgulă mobilă nu memorează semnul lui c. Ne vom referi concret la IBM 360/30, prezentat

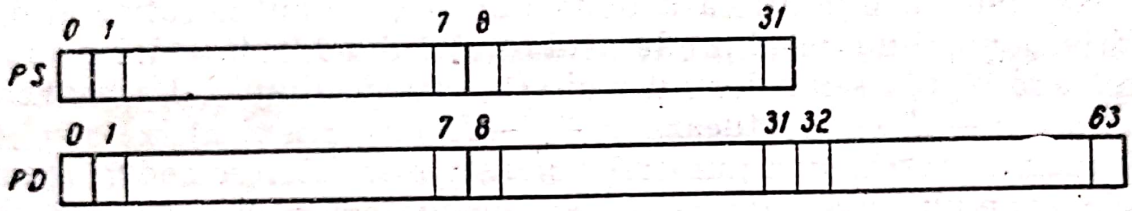


Fig. 7.

în § 1.2.6, care are registre de 32 biți pentru *precizia simplă* (PS) și registre de 64 biți pentru *precizia dublă* (PD). Baza de numerație este $b = 16$. Registrele PS și PD arată ca în figura 7.

Biții 0 din PS și PD memorează semnul lui a. Caracteristica c poate avea semnul + sau —, dar nu este rezervat loc pentru semn, pentru că totdeauna în biții 1—7 se memorează valoarea *nenegativă* $p = c + 64$. Cea mai mică valoare nenegativă p este 0, iar cea mai mare este $2^7 - 1 = 127$. Atunci, c poate varia în domeniul $-64 \leqslant c \leqslant 63$.

În restul de 24, respectiv 56 de biți, se memorează mantisa m, în forma hexazecimal — binară, adică fiecare cifră hexazecimală se memorează în scrierea binară cu patru biți. *Precizia* care se poate realiza este deci de $\frac{24}{4} = 6$ cifre hexazecimale în *PS*, ceea ce înseamnă 7 cifre zecimale, iar în *PD* $\frac{56}{4} = 14$ cifre hexazecimale, adică 16 cifre zecimale; (exact, înseamnă 7,2, respectiv 16,8, cifre zecimale).

Deci, pentru *PS*, $16^{-1} \leqslant |m| \leqslant 1 - 16^{-6}$, iar pentru *PD*, $16^{-1} \leqslant |m| \leqslant 1 - 16^{-14}$. Atunci, numerele reale care pot fi folosite în calcule, la acest calculator, sînt pentru *PS*:

$$16^{-64} \times 16^{-1} = 16^{-65} \leqslant |a| \leqslant 16^{63} \times (1 - 16^{-6}),$$

iar pentru *PD*:

$$16^{-64} \times 16^{-1} = 16^{-65} \leqslant |a| \leqslant 16^{63} \times (1 - 16^{-14}).$$

Scanned with OKEN Scanner

Se vede că marginile din dreapta la PS și PD diferă foarte puțin, începînd cu a șaptea cifră hexazecimală după virgulă. Menționăm că marginile de mai sus, relative la baza 16, sînt riguros exacte. Pentru a ne face o idee mai clară despre magnitudinea numerelor care pot fi folosite în calcule, revenim la baza 10. Reținînd din cele două margini din dreapta pe cea mai mică, anume $16^{63} \times (1 - 16^{-6})$, obținem aproximativ $5{,}4 \times 10^{-79} \leqslant |a| \leqslant 7{,}2 \times 10^{75}$. Acest domeniu, practic, acoperă toate calculele care se cer a fi executate.

Orice calculator efectuează calcule cu mărimi algebrice. Pentru ca aceste calcule să fie posibile, semnul este tratat ca un număr binar. De exemplu, la înmulțirea a două numere, primul negativ, al doilea pozitiv, semnul minus al produsului rezultă din $1 \times 0 = 1$. La adunare și scădere regula semnelor nu mai este atît de simplă. Dacă avem de efectuat $a - d$ se efectuează $a + (-d)$.

Pentru reprezentarea numerelor negative se folosesc coduri speciale, ca de exemplu *codul invers*, *codul complementar*, etc. Un calculator folosește numai unul din aceste coduri. Codul invers sau codul complementar al unui număr se obține automat, fără intervenția programatorului. O prezentare detaliată a acestor coduri se găsește în [10], [6].

1.3.3. Structura unei instrucțiuni. Tipuri de instrucțiuni

Pentru ca un calculator să execute cel mai simplu calcul sau șir de calcule, trebuie să i se indice pas cu pas ce are de făcut. Indicarea se face prin așa-numitele *instrucțiuni mașină*. Instrucțiunea mașină este, deci, unitatea de bază a unui program. Instrucțiunile, pe care calculatorul este capabil să le recunoască, formează *limbajul intern* al acestuia. Limbajul intern se deosebește de la calculator la calculator, după cum limba vorbită se deosebește de la un popor la altul. „Vorbirea", și deci „colaborarea" cu un calculator, se poate face numai în limbajul pe care acesta îl înțelege. (Vom vedea mai departe că un om nu trebuie să cunoască atîtea limbaje interne cîte calculatoare vrea să utilizeze).

Oricît de diferite sînt instrucțiunile mașină de la un calculator la altul, se pot desprinde totuși unele elemente generale. Orice instrucțiune trebuie să conțină un *cod al operației* care urmează să fie executată și cel puțin o *adresă**. Codul operației poate indica adunarea, scăderea, înmulțirea, împărțirea, compararea, deplasarea la stînga, deplasarea la dreapta, citirea din memorie, depunerea în memorie,

* Există totuși instrucțiuni fără adresă și anume instrucțiunea de neoperare și unele instrucțiuni de intrare/ieșire.

Scanned with OKEN Scanner

citirea de pe un mediu suport și depunerea în memorie, etc. Adresa indică locul de unde se ia informația care trebuie prelucrată, sau locul unde trebuie depus rezultatul. Pentru codul operației la fiecare calculator există stabilită o corespondență biunivocă între operația ce trebuie executată și reprezentarea ei formală. De exemplu, 01 însemnează citirea, 02 însemnează adunarea, 03 — scăderea, etc., sau adunarea poate fi notată prin *A*, scăderea prin *S*, etc. Acest ultim mod de reprezentare — cu *A*, *S*, etc. — este un *cod mnemonic* care ajută pe programator să țină minte mai ușor sensul reprezentării formale.

Adresa se reprezintă de asemenea formal, prin cifre, litere sau cifre și litere.

După numărul adreselor dintr-o instrucțiune, există: *instrucțiuni cu o adresă, instrucțiuni cu două adrese, instrucțiuni cu trei adrese* (teoretic, pot exista instrucțiuni cu oricît de multe adrese).

Libajul intern al calculatoarelor care folosesc instrucțiuni cu o adresă este relativ sărac: citire din memorie, depunere în memorie, operații aritmetice, deplasare la stînga, deplasare la dreapta, salt condiționat sau necondiționat la o altă instrucțiune. Instrucțiunea este de obicei asociată cu un *registru acumulator* unde se fac calculele. Atunci, adresa din instrucțiunea de citire indică locul din memorie de unde se aduce în registrul acumulator un anumit număr. La instrucțiunea de adunare, adresa indică locul din memorie de unde se aduce numărul care va fi adunat cu numărul existent în registrul acumulator. Rezultatul adunării rămîne în registrul acumulator. Depunerea lui la o anumită adresă se face prin instrucțiunea de depunere.

La calculatoarele care folosesc instrucțiuni cu două adrese, limbajul intern este mai bogat, iar rapiditatea în execuție este în general mai mare. În mod concret, ne vom referi la IBM 360/30 care folosește instrucțiuni cu două adrese. Forma generală a unei astfel de instrucțiuni, cu indicarea numărului de biți pentru fiecare cîmp, este cea din figura 8.

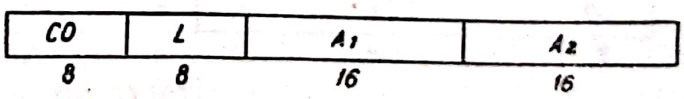


Fig. 8.

CO este *codul operației*, care indică:

— ce operație trebuie executată,

— dacă datele prelucrate sînt în formă zecimală sau binară,

— dacă datele sînt în registre sau în memoria principală,

— cît de lungă este instrucțiunea,

— care este lungimea operanzilor, atunci cînd *L* lipsește,

— dacă prelucrarea este în virgulă fixă, în virgulă mobilă sau zecimală.

Scanned with OKEN Scanner

L indică *lungimea* cîmpurilor prelucrate, pentru informaţia luată din memoria principală. Dacă cele două cîmpuri de la adresele $A1$ şi $A2$ au lungimi diferite, atunci L se descompune în $L1$ şi $L2$, fiecare din aceste cîmpuri avînd cîte patru biţi. L poate să lipsească din instrucţiunile care prelucrează informaţii de lungime fixă (standard).

$A1$ este *adresa primului operand* şi indică *de unde* se ia primul operand şi *unde* se vor depune rezultatele operaţiei indicate de CO. Deci operaţia de depunere, care este o operaţie separată în instrucţiunea cu o adresă, este conţinută în instrucţiunea cu două adrese.

$A2$ este *adresa celui de al doilea operand* şi indică *de unde* se ia operandul al doilea.

Prelucrarea datelor în binar se face în registre. Există 16 registre, numerotate 0—15, pentru virgula fixă, fiecare registru avînd 32 biţi, adică patru baiţi. Pentru prelucrarea în binar, în virgulă mobilă există patru registre, fiecare de cîte 64 biţi. Ele sînt numerotate 0, 2, 4, 6.

Pentru adresarea unui cîmp de baiţi din memoria principală, se indică adresa primului bait al cîmpului şi numărul de baiţi ($A1$, $L1$ şi $A2$, $L2$). Atunci, din forma instrucţiunii indicate mai sus, se deduce că nu se pot adresa cîmpuri al căror prim bait are o adresă mai mare de $2^{16} - 1 = 65\,535$, pentru că această adresă nu începe în spaţiul rezervat lui $A1$ sau $A2$.

Aces inconvenient se evită prin *calcularea adresei cîmpului adresat*, operaţie pe care o face automat calculatorul. În acest caz, instrucţiunea are forma din figura 9.

$B1$ şi $B2$ se numesc *registre de bază*. Numerele binare ce se pot înscrie în lungimile afectate sînt 0—15. $D1$ şi $D2$ se numesc *deplasări* de la conţinutul registrului de bază.

Adresa cîmpului adresat se obţine din conţinutul registrului de bază + deplasarea, adică $A1 = [B1]^* + D1$, $A2 = [B2] + D2$. Se poate ca $B1$ şi $B2$ să aibă acelaşi conţinut.

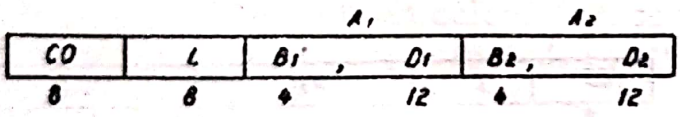


Fig. 9.

Se vede că numerele înscrise în $D1$ şi $D2$ nu pot depăşi $2^{12} - 1 = 4095$ baiţi, iar numerele înscrise într-unul din registrele 0 — 15 nu pot depăşi $2^{32} - 1$, ceea ce înseamnă un număr de ordinul miliardelor de baiţi.

Fie, de exemplu, $B1 = 10$; iar conţinutul lui 10, adică $[10] = 80750$. Fie $D1 = 500$. Atunci, $A1 = [10] + 500 = 81250$. Deci,

---

* Aici, am notat prin [ ] conţinutul registrului de bază.

Scanned with OKEN Scanner

se poate adresa un cîmp de baiţi care începe la adresa 81250, care altfel nu ar fi încăput în $A1$, folosind prima formă a instrucţiunii.

S-au mărit deci posibilităţile de adresare şi s-a obţinut posibilitatea deplasării unui program în memoria principală.

Mărirea posibilităţilor de adresare se mai poate face prin folosirea a încă un registru X, numit *registru index*. El se foloseşte numai pentru $A2$ şi anume $A2 = [X2] + [B2] + D2$.

Din această prezentare a instrucţiunii cu două adrese se întrevede posibilitatea extinderii unui sistem de calcul prin înlocuirea unităţii centrale cu o unitate centrală din aceeaşi familie, dar, de exemplu, cu o memorie internă mai mare.

Calculatorul IBM 360/30, prezentat în § 1.2.6, are o memorie internă de $64K$ baiţi, unde $K = 1024$. Extinderea memoriei la $128K$ este compatibilă cu forma instrucţiunii care foloseşte adresarea prin registrul de bază sau prin registrul index.

Pentru a prezenta instrucţiunea cu trei adrese, ne vom referi la calculatorul SEA 3900. Notăm adresele cu $A1$, $A2$, $A3$. La operaţiile aritmetice, $A1$ şi $A2$ sînt adresele operanzilor, iar $A3$ este adresa rezultatului. La instrucţiunile de salt condiţionat $A1$ şi $A2$ sînt adresele termenilor care se compară, iar $A3$ este adresa instrucţiunii care trebuie executată, dacă saltul are loc. La citirea din memorie şi scrierea pe bandă magnetică sau la imprimantă, $A1$ este adresa primului caracter din memoria principală, $A2$ este adresa externă, iar $A3$ este adresa unde se memorează adresa $A1$ şi $A2$ în memoria principală, în caz de eroare.

1.4. POSIBILITĂŢI DE LUCRU ALE UNUI SISTEM DE CALCUL

1.4.1. Posibilităţi fizice

Descrierea completă, într-un singur paragraf, a posibilităţilor fizice ale sistemelor de calcul, fără a ne referi la un sistem concret, este imposibilă. De aceea, vom prezenta criterii general valabile pentru orice sistem, cu unele exemplificări concrete pentru anumite sisteme de calcul.

În primul rînd, posibilităţile fizice sînt funcţie de configuraţia de calcul şi anume de:

— tipul unităţii centrale;
— prezenţa sau lipsa memoriilor tampon;
— lipsa canalelor sau existenţa unui anumit număr de canale;

Scanned with OKEN Scanner

— prezența sau lipsa memoriilor externe;
— capacitatea și viteza de lucru a echipamentului periferic.

Tipul unității centrale este important din mai multe puncte de vedere, dintre care amintim: capacitatea memoriei interne, timpul de acces la o adresă a memoriei interne și viteza cu care dispozitivul aritmetic — logic efectuează operațiile. Am văzut că memoria internă este formată din celule de memorie, fiecare celulă avînd o adresă. Numărul de biți din care este formată o celulă, în general, diferă de la o unitate centrală la alta. Celulele de memorie se mai numesc uneori *unități adresabile ale memoriei* sau *cuvinte de memorie.*

De exemplu, calculatorul Bull Gamma 60 are o memorie pînă la 32768 cuvinte, fiecare cuvînt avînd 24 biți. Unitatea centrală 7302 a calculatorului IBM 7094 conține 32768 cuvinte, fiecare cuvînt avînd 36 biți. Seria Elliott 4100 are cuvinte de 24 biți. Pentru acest calculator, unitatea centrală 4120 poate avea pînă la 32768 cuvinte, iar unitatea centrală 4130 poate avea pînă la 262144 cuvinte. Seria ICT 1900 are memorii interne care variază între 4096 și 262144 cuvinte de 24 biți.

La seria IBM 360 baitul (8 biți) este cea mai mică unitate de informație care poate fi adresată. Unitățile centrale au memorii interne care, funcție de model, pot varia între următoarele limite (socotite în baiți): model 30—8 la 64 *K*; model 40—16 la 256 *K*; model 50—64 la 256 *K*; model 60—128 la 512*K*; model 70—256 la 512 *K*; model 90—512 *K*.

Timpul de acces la o adresă este o caracteristică importantă a memoriilor interne. Acest timp, la calculatoarele actuale, se măsoară în micro-secunde (μs, $\mu = 0{,}000\,001$ s $= 10^{-6}$s). La calculatorul Bull Gamma 60, timpul necesar unei depuneri la o adresă, sau timpul necesar unei citiri de la o adresă este de 10 μs. Acesta reprezintă *ciclul memoriei interne.*

Ciclul memoriei interne la IBM 7094 cuprinde în mod obligatoriu două faze: citirea și depunerea și el este de 2μs.

Seria Elliott 1400 oferă două clase de memorii interne care au ciclul memoriei citire și depunere, de 6 μs, respectiv 2μs.

Acest ciclu, la seria IBM 360, funcție de model, se prezintă astfel: 2μs la modelele 30, 50 și 60; 2,5 μs la modelul 40; 1 μs la modelele 62 și 70.

Viteza cu care dispozitivul aritmetic — logic efectuează operațiile din limbajul intern al calculatorului variază de asemenea de la un tip de unitate centrală la altul.

Se indică, de obicei, viteza de executare a operațiilor aritmetice. Ea este privită diferențiat pentru prelucrarea în virgulă fixă (*VF*) și pentru prelucrarea în virgulă mobil (*VM*) și se exprimă în μs/operație.

Scanned with OKEN Scanner

La IBM 7094, viteza în virgulă fixă este de: 3,2 μs la adunare şi scădere; 9,2 μs la înmulţire şi 15,2 μs la împărţire, iar în virgulă mobilă: 5,2 μs la adunare şi scădere; 9,2 μs la înmulţire şi 17,2 μs la împărţire.

În tabelele ce urmează sînt prezentate, după [10], vitezele de lucru pentru Elliott 4100, ICT 1900, IBM 360.

TABELUL 3

Viteza pentru seria Elliott 4100

Viteza \ Modelul		4120		4130	
		6μs	2μs	6μs	2μs
VF	Adunare	12	6	12	4,5
	Înmulţire	87	81	22	15
VM	Adunare	340	235	25	14
	Înmulţire	678	477	55	45

TABELUL 4

Viteza şi ciclul memoriei pentru seria ICT 1900

Caracteristica \ Modelul		1902	1903	1904/1905	1906/1907	1909
Ciclul memoriei (μs)		6	2	2	2,25 sau 1,25	6
VF	Adunare	18	7	7	2,5 pentru 1,25	18
	Înmulţire	1500	650	40	11,5	67
VM	Înmulţire	5250	2250	285/29	60/7,75	37

TABELUL 5

Viteza şi ciclul memoriei pentru seria IBM 360

Caracteristica \ Modelul		30	40	50	60	62	70
Ciclul memoriei (μs)		2	2,5	2	2	1	1
VF	Adunare	29	7,5	3,3	1	0,9	0,4
	Înmulţire	303	83	28	5,8	5,6	2,7
VM	Înmulţire	312	77	21	5,3	5,1	2,2

Scanned with OKEN Scanner

Din aceste tabele se observă că există calculatoare care efectuează 2,5 — 4 milioane de adunări/s. Pentru viitor, sînt anunțate (CDC, IBM) viteze de 8—20 milioane de adunări/s.

1.4.2. Posibilități de programare; limbaje algoritmice și compilare

După ce un sistem de calcul a fost construit și instalat, urmează exploatarea lui. Exploatarea se face prin *programarea* sistemului de a executa problemele dorite de utilizator. Programarea constă în transpunerea problemei într-o succesiune precisă de operații, pe care le poate recunoaște și executa calculatorul. Succesiunea de operații este aleasă din limbajul intern al calculatorului și formează *programarea în cod* sau, altfel spus, *programarea în limbajul intern*. Cu cît limbajul intern este mai elastic și mai bogat, cu atît programarea, adică scrierea programului de rezolvare a unei probleme, se face mai ușor. Dar, un limbaj intern foarte elastic și foarte bogat duce la complicarea din punct de vedere fizic a unității centrale, iar un dispozitiv aritmetic-logic foarte simplu duce la complicarea posibilităților de programare. De aceea, construirea unui calculator se face concomitent cu studierea posibilităților sale de programare.

Din cele spuse pînă acum rezultă că, pentru ca un utilizator să scrie programe pentru mai multe calculatoare diferite, el trebuie să cunoască tot atîtea limbaje interne de programare. Acesta este un inconvenient, dar nu este cel mai mare. Oricît de elastic și de bogat ar fi limbajul intern, programarea în cod este dificilă, supusă multor greșeli și ea necesită adesea un volum mare de muncă, chiar pentru o problemă simplă, munca de programare depășind chiar efortul de rezolvare directă a problemei.

Toate problemele trebuie să fie privite în desfășurarea lor cronologică. La începutul utilizării calculatoarelor, se folosea numai programarea în cod, ca unică modalitate de lucru. Astăzi posibilitățile de programare au crescut prin folosirea *limbajelor algoritmice de programare*. Limbajul algoritmic de programare este un limbaj artificial, construit cu o anume structură gramaticală și care folosește litere, cifre și un număr de semne speciale, pentru exprimarea dorințelor utilizatorului, adică pentru scrierea programului. Elementele sale de bază nu depind de un anumit calculator, ci sînt general valabile pentru orice calculator. Dar, am spus că orice calculator nu poate executa o problemă decît dacă ea îi este furnizată în limbajul său intern. Apare deci necesitatea traducerii programului scris dintr-un limbaj algoritmic — numit *program sursă* — într-un program în limbajul intern — numit *program obiect*.

Scanned with OKEN Scanner

Traducerea programului sursă în program obiect o face un program special, scris în limbajul intern al acelui calculator pe care îl folosim în rezolvarea problemei concrete şi care se numeşte *program de translatare* sau *program de compilare* sau, mai simplu, *translator* sau *compilator*.

Translatorul este un program laborios, cu cîteva mii de instrucţiuni maşină, dar care este construit o dată pentru totdeauna pentru un calculator.

Un program sursă scris într-un anume limbaj algoritmic este valabil pentru orice calculator, care are un program de translatare pentru acel limbaj.

Există calculatoare care posedă translatoare pentru mai multe limbaje algoritmice. Programarea într-un limbaj algoritmic nu necesită cunoaşterea structurii interne a calculatorului.

Limbajul algoritmic propriu-zis nu conţine instrucţiunile de intrare/ieşire. Aceste instrucţiuni, şi numai acestea, sînt specifice fiecărui calculator în parte.

În prezent se cunosc numeroase limbaje algoritmice, dintre care cele mai des utilizate sînt: ALGOL (ALGOrithmic Language), FORTRAN (FORmula TRANslating)*, COBOL (COmmon Business Oriented Language), PL/1 (Programming Language 1).

Primele încercări de folosire a unor limbaje algoritmice s-au făcut prin 1951—1952. În 1957 a apărut o primă versiune a limbajului FORTRAN şi s-a construit compilatorul corespunzător, pentru IBM 704. În decembrie 1959 a fost încheiată prima dare de seamă referitoare la COBOL şi publicată în S.U.A. în 1960. Un raport complet asupra limbajului ALGOL apare în 1960, în urma conferinţei de la Paris. Limbajul PL/1 este un limbaj algoritmic de dată mai recentă.

Limbajele ALGOL, FORTRAN şi PL/1 sînt destinate problemelor matematice şi de cercetare ştiinţifică. Ele uşurează mult munca de programare, permiţînd scrierea formulelor şi a expresiilor într-o formă apropiată de scrierea matematică.

Limbajul COBOL este destinat problemelor economice, care constau din calcule relativ simple, dar care cer tabelări de rezultate în forme complicate.

Programele de translatare ale unui sistem de calcul fac parte dintr-un sistem de programe de comandă şi control, numit *sistem de operare*. Fiecare sistem de calcul are un sistem de operare propriu, care face

---

* FORTRAN este de fapt o prescurtare a denumirii „Mathematical Formula Translating System" prin FORmula TRANslation.

Scanned with OKEN Scanner

în mod automat legătura funcţională între toate elementele configuraţiei de calcul. Ca şi configuraţia de calcul, sistemul de operare este de *natură modulară* adică poate fi îmbogăţit şi îmbunătăţit pentru optimizarea folosirii resurselor de calcul.

1.5. DESPRE SISTEMUL DE OPERARE AL UNUI SISTEM DE CALCUL

1.5.1. Ce este un sistem de operare şi necesitatea folosirii lui

Toate dispozitivele fizice ale unui sistem de calcul sînt inerte cît timp nu intervine programul care să facă legătura funcţională între ele. La primele calculatoare acest program era chiar programul de rezolvare a unei probleme, scris de utilizator. Acesta de obicei făcea legătura dintre un dispozitiv de intrare (de exemplu cititorul de bandă perforată), unitatea centrală şi un dispozitiv de ieşire (de exemplu o maşină de scris electrică pentru tipărirea rezultatelor). După ce programul unui utilizator era executat, o persoană calificată pentru mînuirea calculatorului numită *operator* pregătea pentru execuţie (punea banda ce conţinea programul în cititorul de bandă şi acţiona butonul de începere a citirii) un alt program, care realiza o nouă legătură funcţională între dispozitivele de intrare/ieşire şi prelucrare.

La sistemele de calcul actuale, legătura funcţională este făcută de un sistem de programe, numit *sistem de operare*. El este scris de constructorii de sisteme de operare în limbajul intern al calculatorului, livrat odată cu cumpărarea calculatorului şi construit odată cu construirea acestuia.

Spus pe scurt, sistemul de operare a apărut datorită necesităţii optimizării folosirii resurselor fizice (hardware) şi a resurselor de programare (software) ale unui sistem de calcul. Sistemul de operare este de natură modulară, adică poate fi îmbogăţit şi îmbunătăţit conform dorinţelor utilizatorilor sistemului de calcul respectiv. Deci, o configuraţie de calcul dată poate avea mai multe sisteme de operare, dar, într-un anumit interval de timp, se foloseşte numai unul singur. Evident, modificări în configuraţia de calcul impun modificări corespunzătoare şi în sistemul de operare. Acel capitol al programării calculatoarelor care se ocupă cu conceperea sistemelor de operare şi a limbajelor formale se numeşte *software de bază*.

Scanned with OKEN Scanner

Calculatorul la care însuşi programul utilizatorului face legătura funcţională între dispozitivele fizice, practic, nu are sistem de operare. Nu înseamnă însă că sistemul de calcul care are sistem de operare nu are operator; are operator cu îndatoriri noi.

Să enumerăm acum o parte din avantajele şi necesităţile folosirii unui sistem de operare: 1. În primul rînd, numai un sistem de operare poate să urmărească desfăşurarea rapidă a executării programului utilizatorului. Dispozitivul de comandă al unităţii centrale lucrează împreună cu sistemul de operare. 2. Întocmirea automată a programului de canal pentru transferul de informaţie de la unitatea de intrare sau spre unitatea de ieşire se face sub controlul sistemului de operare. 3. Compilarea programului sursă în program obiect se face de un translator, care este o parte a sistemului de operare. 4. Prin compilare se obţine programul obiect în adrese relative, iar pentru executare este necesară alocarea de adrese absolute şi încărcarea lui în memoria internă. Aceste operaţii le fac nişte programe specializate ale sistemului de operare. 5. Programul sursă, scris de utilizator, poate să conţină instrucţiuni care nu corespund sintaxei limbajului algoritmic folosit (greşeli de sintaxă). Găsirea şi indicarea greşelilor sintactice din program o face sistemul de operare, prin intermediul compilatorului. 6. De asemenea, el verifică corectitudinea funcţionării tuturor dispozitivelor fizice şi semnalează erorile care apar. 7. Sistemul de operare activează automat un nou program, care se află în cititor, la terminarea executării programului anterior. La calculatoarele fără sistem de operare, activarea unui nou program este făcută de operator. 8. Asigură protecţia memoriei principale împotriva distrugerii programului memorat care se execută. 9. Sistemul de operare asigură păstrarea informaţiei în memoriile externe. 10. Permite exploatarea sistemului de calcul în *regim multiplex* (time sharing), sau în *regim de multiprogramare* (multiprogramming), sau în *regim de multiprelucrare* (multiprocessing), sau în *regim de teleprelucrare* (teleprocessing). (O descriere a acestor moduri de lucru se va face în § 1.6.) Exploatarea sistemului numai într-unul din regimurile enumerate, sau în mai multe, este funcţie de tipul unităţii centrale. 11. Sistemul de operare transmite mesaje operatorului şi primeşte răspunsuri şi/sau comenzi prin intermediul consolei unităţii centrale. 12. Menţine împărţirea într-o anume formă a memoriei principale. (Alte avantaje şi necesităţi le vom vedea la descrierea unui sistem de operare concret.)

Sistemul de operare al unui sistem de calcul se află memorat pe un disc sau pe o bandă şi formează *rezidenţa sistemului*. Variante diferite ale aceluiaşi tip de sisteme de operare conduc la rezidenţe diferite. Într-un interval de timp dat, sistemul de calcul foloseşte o singură rezidenţă (detalii se vor vedea în § 1.5.4). Crearea rezidenţei sistemului

Scanned with OKEN Scanner

pentru o configurație de calcul se numește *generarea sistemului de operare a configurației*.

Cumpărarea unei configurații de calcul se face prin plătirea separată a echipamentului fizic (hardware) și a sistemului de operare, cu biblioteca de subprograme (software). În decursul timpului, ponderea costului software-ului, relativ la costul total (hardware + software), a evoluat astfel : 30% la primele sisteme de calcul cu sistem de operare, apoi 50% cu cîțiva ani înainte, iar în prezent ponderea este de 65%—70%, cu tendința de a crește în continuare.

În cele ce urmează, ne vom ocupa numai de calculatoarele care folosesc sistem de operare.

1.5.2. Protejarea împărțirii memoriei principale și bibliotecile rezidenței sistemului

La memoria principală se disting două părți : o parte necesară activităților sistemului de operare (a rezidenței) și o parte necesară executării programelor utilizatorilor. Pentru ca programele din cele două părți să nu se suprapună, sau să nu interfereze în timpul funcționării, trebuie asigurată protejarea împărțirii memoriei principale. Totuși protecția este opțională, adică poate să fie instalată sau să nu fie instalată la o configurație de calcul. Instalarea protecției necesită amenajări fizice (hardware) suplimentare, care atrag modificări specifice și în rezidența sistemului. În general, protecția se instalează.

Protecția instalată poate avea unul din următoarele două tipuri : *protecția la depunere* incorectă sau *protecția la depunere și citire* incorectă. Nu se instalează numai protecție la citire incorectă. Protecția la depunere se mai numește protecția împotriva distrugerii conținutului memoriei, iar protecția la depunere și citire se mai numește protecție împotriva folosirii incorecte a memoriei.

La sistemul IBM 360, pentru asigurarea protecției, memoria este împărțită în blocuri de cîte $2K$ baiți. Fiecare bloc are asociat un cîmp de cinci biți. Bitul 5, prin conținutul său, indică tipul protecției și anume : 0 indică protecția la depunere și citire, iar 1 indică protecția la depunere. Primii patru biți din cei cinci conțin un cod care formează *cheia blocului*. Aceeași cheie poate să fie folosită la un bloc sau la mai multe blocuri. Aceasta înseamnă că memoria principală poate să fie împărțită în domenii distincte (cu chei distincte), cu capacități de $0K$, 2k, $4K$, $6K$, ... baiți.

În această accepție, la IBM 360/30, descris în § 1.2.6, împărțirea memoriei principale este următoarea :

Scanned with OKEN Scanner

a) Domeniul sistemului de operare (în § 1.5.4. vom vedea că este domeniul supervizorului şi al ariei de tranziţie), cu cheia de protecţie 0000.

b) Domeniul *BG* (Background), cu cheia 0001.

c) Domeniul *F2* (Foreground 2), cu cheia 0010.

d) Domeniul *F1* (Foreground 1), cu cheia 0011.

BG, *F2* şi *F1* sînt domeniile în care se execută programele utilizatorilor, sub controlul sistemului de operare. În cele trei domenii se pot executa simultan trei programe independente.

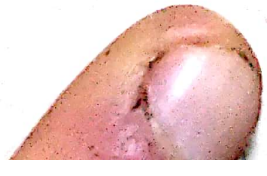
Să urmărim acum cum este asigurată protecţia domeniilor sau a partiţiilor menţionate. Cheia domeniilor este pusă de sistemul de operare în momentul alocării memoriei pentru fiecare partiţie. Mecanismul accesului la o partiţie (ne referim numai la *BG*, *F2* sau *F1* de care este interesat utilizatorul) se bazează pe faptul că atît programul cît şi datele programului trebuie să se afle în acelaşi domeniu. Programul obiect este depus, instrucţiune cu instrucţiune, într-o partiţie a memoriei specificată de utilizator. La execuţie, există un cîmp de opt baiţi, numit cîmpul *PSW* (Program Status Word), care conţine informaţii despre instrucţiunea curentă care se execută. Deci, conţinutul cîmpului *PSW* se reactualizează cu fiecare instrucţiune din program, în momentul executării ei. În *PSW* există un cîmp de patru biţi, care formează *cheia de protecţie din PSW*. Ea este pusă de unitatea centrală şi este chiar cheia partiţiei în care se află instrucţiunea curentă. Prin adresele sale *A1* şi *A2*, instrucţiunea adresează operanzii (datele asupra cărora acţionează codul operaţiei). Dacă cheia partiţiei în care se află operanzii nu coincide cu cheia de protecţie din *PSW*, înseamnă că s-au adresat operanzi din afara partiţiei rezervate programului care se execută.

La violarea protecţiei prin depunere în afara partiţiei, operaţia de depunere este suprimată, dar programul continuă. La violarea protecţiei prin citire incorectă, programul este totdeauna anulat.

Utilizatorul nu trebuie să cunoască cheia partiţiilor, dar trebuie să fie sigur că programul său începe în partiţia a cărei capacitate el însuşi a indicat-o.

Accesul la conţinutul cheii partiţiilor se face prin instrucţiunile Assembler *SSK* (SET STORAGE KEY) şi *ISK* (INSERT STORAGE KEY). Prin *SSK* se stabileşte cheia partiţiei, conform dorinţelor utilizatorului, iar prin *ISK* se poate afla cheia partiţiei, stabilită de unitatea centrală.

Am văzut că orice calculator execută numai acele programe care sînt scrise sau sînt compilate în limbajul său intern. Dar prin compilare sau scriere directă în limbaj intern se obţin adrese relative pentru instrucţiuni şi pentru date. Executarea este posibilă numai dacă se



Scanned with OKEN Scanner

trece programul din adrese relative în adrese absolute. Această operație complexă o face sistemul de operare. Relativitatea adreselor se referă la partițiile memoriei.

Din cele de mai sus deducem că există trei categorii de programe: programe-sursă, scrise într-un limbaj algoritmic, programe-obiect în adrese relative, ambele neexecutabile în aceste forme, și programe obiect în adrese absolute, deci executabile.

Conform acestor categorii de programe, un sistem de operare poate folosi trei tipuri de *biblioteci ale sistemului*. Pe lîngă acestea, mai pot exista *biblioteci private ale utilizatorului*. Bibliotecile sistemului se află în rezistență, iar bibliotecile private se află în memoriile externe. Numărul bibliotecilor private este teoretic nelimitat.

Vom descrie aceste biblioteci pentru sistemul IBM 360. Pot exista trei biblioteci ale sistemului și anume:

1. Biblioteca programelor-obiect, în adrese absolute, în fază executabilă sau biblioteca *CIL* (Core Image Library).
2. Biblioteca programelor-obiect, în adrese relative, neexecutabile sau biblioteca *RL* (Relocatable Library).
3. Biblioteca programelor-sursă, neexecutabile sau biblioteca *SSL* (Source Statement Library).

Biblioteca *CIL* este obligatorie la orice rezidență. Bibliotecile *RL* și *SSL* sînt opționale, funcție de dorințele utilizatorilor sistemului de calcul. Existența lor este menționată în momentul generării sistemului de operare. Capacitățile de memorare a bibliotecilor sistemului sînt indicate tot în momentul generării. Spațiul alocat este funcție de mărimea programelor și de numărul lor.

Depunerea programelor într-una din bibliotecile sistemului se numește *catalogare*. Programului catalogat i se dă un nume, cu ajutorul căruia el este apelat spre a fi executat. Deci numele, sub care se cataloghează programele, trebuie să fie distincte.

În *CIL* catalogarea poate fi *temporară sau permanentă*. Programul catalogat temporar se șterge automat din bibliotecă, după ce el a fost executat. La catalogarea permanentă el rămîne în bibliotecă cît timp dorește utilizatorul.

În *RL* și *SSL* catalogarea este numai permanentă. Programele rămîn valabile pînă cînd se dă comanda de ștergere.

Bibliotecile private pot fi de tip *RL* sau de tip *SSL*. Catalogarea programelor în aceste biblioteci este permanentă. Folosirea lor o vom vedea în § 1.7.

Scanned with OKEN Scanner

1.5.3. Numele simbolice ale unităţilor de intrare/ieşire

Am văzut că, prin folosirea unui limbaj algoritmic de programare, se asigură independenţa scrierii programului de limbajul intern al calculatorului. Dar programul sursă sub formă de program obiect şi datele problemei, în final trebuie să ajungă în memoria principală. Apoi rezultatele calculelor trebuie extrase şi, cel mai adesea, tipărite. Deci programul utilizatorului devine dependent de unităţile de intrare/ieşire. În consecinţă, programul sursă ar trebui să indice tipul perifericului utilizat la un moment dat şi *adresa* sa în configuraţia concretă care execută programul. Atunci ar însemna că, dacă utilizatorul nu cunoaşte adresele perifericelor configuraţiei, nu poate să scrie un program care să fie executat. Înlăturarea acestui inconvenient şi deci asigurarea independenţei programului sursă de unităţile de intrare/ieşire se fac prin folosirea unor *nume simbolice* asociate unităţilor de intrare/ieşire. Prin aceasta se asigură şi universalitatea programului sursă pentru orice altă configuraţie compatibilă.

Numele simbolice desemnează în mod fictiv orice periferic. Se mai spune că ele reprezintă *unităţi logice*. Pentru un sistem de calcul, ele sînt luate dintr-o mulţime de nume simbolice, finită şi fixată. De exemplu, prin *SYSRDR* ne referim la cititorul de cartele care, în cadrul configuraţiei de calcul, are o adresă care nu ne este cunoscută *în timpul scrierii* programului sursă. Dar *la execuţie*, unităţii centrale trebuie să i se comunice ce a înţeles utilizatorul prin numele simbolic respectiv. Această comunicare se face prin *asignarea* unei unităţi fizice concrete numelui simbolic folosit. Asignarea se face *înainte* de executarea programului, prin *comenzi de asignare*. Numărul acestora este egal cu numărul numelor simbolice folosite. Dacă programul sursă este perforat pe cartele, comenzile de asignare se perforează tot pe cartele şi se pun în faţa pachetului sursă. Aceste cartele pot fi întocmite de operatorul configuraţiei, prin consultarea listei de semnificaţii pe care o întocmeşte utilizatorul.

Asignările sînt de două feluri: *permanente* şi *temporare*. Asignările permanente se fac la generarea sistemului sau prin comenzi de asignare permanentă. Ele rămîn valabile pînă la dezasignarea lor prin comenzi de dezasignare, date prin cartele sau prin consola unităţii centrale. Asignările de la generare se numesc *asignări standard*.

În tabelul 6 indicăm numele simbolice folosite de IBM 360/30.

Primele nouă nume se numesc *nume simbolice ale sistemului de calcul*. Numele *SYS*000 — *SYS* max se numesc *nume simbolice ale utilizatorului*. Prin max am notat numărul maxim pînă la care poate să ajungă aceste nume. Numărul este stabilit în momentul generării sistemului de operare al configuraţiei de calcul.

Scanned with OKEN Scanner

TABELUL 6

Numele simbolice şi semnificaţia lor

Numele simbolic	Semnificaţia fizică	Semnificaţia logică
SYSRDR	Cititor de cartele Bandă Disc	Reprezintă unitatea de intrare a instrucţiunilor de comandă pentru Job Control.
SYSIPT	Cititor de cartele Bandă Disc	Reprezintă unitatea de intrare a instrucţiunilor programului sursă sau al programului obiect.
SYSPCH	Perforator de cartele Bandă Disc	Reprezintă unitatea de ieşire.
SYSLST	Imprimantă Bandă Disc	Reprezintă unitatea de ieşire.
SYSLNK	Disc	Reprezintă unitatea de intrare/ieşire pentru programul Linkage Editor.
SYSLOG	Consola unităţii centrale	Reprezintă unitatea de intrare/ieşire, unitatea de listare a instrucţiunilor programului Job Control, şi unitatea de comunicare operator — unitatea centrală.
SYSRES	Disc	Rezidenţa configuraţiei de calcul.
SYSRLB	Disc	Biblioteca privată de tip *RL*.
SYSSLB	Disc	Biblioteca privată de tip *SSL*
SYS000 pînă la *SYSmax*	Disc Bandă	Reprezintă orice unitate de disc sau de bandă asignată.

Cînd *SYSRDR* şi *SYSIPT* sînt asignate la aceeaşi unitate de intrare, se poate folosi numele simbolic unic *SYSIN*. Cînd *SYSLST* şi *SYSPCH* sînt asignate la aceeaşi unitate de ieşire, se poate folosi numele simbolic unic *SYSOUT*.

Scanned with OKEN Scanner

Orice rezidenţă pentru o configuraţie IBM 360/30 trebuie să dispună de cel puţin *SYS000—SYS009*.

Asignările standard sînt de obicei următoarele: *SYSIN* — cititorul de cartele; *SYSPCH* — perforatorul automat de cartele; *SYSLST* — imprimanta; *SYSLOG* — consola; *SYSRES* şi *SYSLNK* — unităţi de discuri.

La programarea în FORTRAN, pentru referirea la unele nume logice se folosesc numerele 1—15 şi anume: 1 pentru *SYSIPT*, 2 pentru *SYSPCH*, 3 pentru *SYSLST*, 4—14 pentru *SYS001—SYS011* şi 15 pentru *SYSLOG*. Astfel, dacă datele unui program sînt pe cartele, citirea lor cu cititorul de cartele se face prin instrucţiunea READ(1), iar tipărirea la imprimantă sau la consolă a rezultatelor din memoria principală se face prin WRITE (3), respectiv WRITE (15).

1.5.4. Descrierea şi funcţiunile unui sistem de operare concret

Ne vom referi la sistemul IBM 360. Acest sistem foloseşte pînă în prezent următoarele sisteme de operare:

1. Sistemul de operare de bază — *BPS* (Basic Program Support); care nu foloseşte periferice rapide (discuri şi benzi);
2. Sistemul de operare cu benzi — *TOS* (Tape Operating System), care foloseşte benzi magnetice;
3. Sistemul de operare cu discuri care cuprinde:

a) Sistemul de operare de bază — *BOS* (Basic Operating System);
b) Sistemul de operare — *DOS* (Disk Operating System);
c) Sistemul de operare —*OS* (Operating System).

Limita inferioară a capacităţii în baiţi a memoriei unităţii centrale, pentru sistemul de operare cu discuri, este de 8*K* pentru *BOS*, 16*K* pentru *DOS* şi 64*K* pentru *OS*.

BOS suportă numai unităţi de discuri de tipul 2311, *DOS* suportă unităţi 2311 şi 2321, iar *OS* suportă orice fel de unităţi de discuri.

Sistemele de operare menţionate se includ astfel:

$$BPS \subset TOS \subset BOS \subset DOS \subset OS.$$

Pe lîngă sistemele de operare indicate, IBM 360 mai poate folosi sistemul de operare — *TSS* (Time Sharing System) pentru prelucrarea în regim de divizare a timpului.

Vom descrie acum sistemul de operare *DOS* (folosit de configuraţia descrisă în § 1.2.6). Sistemul de operare *DOS* se compune din:

1. Programe de control: programul supervizor, programul iniţial de încărcare *IPL* (Initial Program Loader), programul Job Control;

Scanned with OKEN Scanner

2. Programe de serviciu: programul editor de legături (Linkage Editor), programul bibliotecar (Librarian);

3. Programe de prelucrare: programele de compilare, programele utilitare, programele de sortare și fuziune (Sort/Merge), programul autotest, (Autotest), programe aplicative.

Există programe de compilare pentru următoarele limbaje: ASSEMBLER, Basic FORTRAN IV, FORTRAN IV, COBOL, RPG, PL/1, GPSS, etc.

În sistemul de operare *DOS* este obligatoriu să intre programele de control, programul Linkage Editor și cel puțin un program de compilare.

Sistemul de operare al calculatoarelor IRIS-50 și FELIX C-256 este aproape identic cu sistemul *DOS* descris anterior. Deosebirile neesențiale constau în faptul că acestea folosesc compilatoare ASSIRIS și LPG, iar funcțiunile programului Job Control sînt îndeplinite de programul monitor.

Dăm acum o descriere succintă a funcțiilor cîtorva programe care fac parte din sistemul de operare *DOS*. Împărțirea memoriei principale și cheile de protecție ale domeniilor se văd în figura 10.

0 Domeniul supervizorului
0 Aria de tranziție
1 Domeniul *BG*
2 Domeniul F_2
3 Domeniul F_1

Fig. 10. Împărțirea memoriei principale.

Programul Supervizor este programul principal de control care conduce activitatea întregului sistem de calcul. El se găsește în *CIL* și conține un nucleu de subprograme care se găsesc permanent în memoria principală atunci cînd se execută programele utilizatorilor sau, altfel spus, cînd unitatea centrală este în *starea activă* și un număr de subprograme care sînt încărcate în aria de tranziție, de pe discul rezidenței, numai atunci cînd este nevoie de ele la executarea unui program. Subprogramele permanente din memorie formează *nucleul supervizorului*. Acesta conține subprograme pentru:

— activitatea canalelor (Channel Scheduler);
— protecția memoriei;
— chemarea supervizorului (Supervizor Call);
— tratarea a cinci feluri de întreruperi;
— cronometrarea timpului;
— încărcarea programelor în memoria principală (System Loader);
— tabloul de control al unităților de intrare/ieșire.

În aria de tranziție, la executarea programelor, cel mai frecvent sînt chemate subprogramele:

— de deschidere și de închidere a unui fișier;

Scanned with OKEN Scanner

— de comunicare cu operatorul, prin consolă;
— de prelucrare a sfîrşitului de program;
— de prelucrare a erorilor programului.

Programul IPL este un program de încărcare. Supervizorul, pentru a putea să îşi execute funcţiunile sale, trebuie încărcat din *SYSRES* de pe disc, în memoria principală. Această încărcare o face programul *IPL*. Dar *IPL* se află tot în *SYSRES* şi deci şi el trebuie activat, adică încărcat în memoria principală . Încărcarea lui se face în două faze: în prima fază, prin apăsarea manuală a cheii *LOAD* de pe panoul unităţii centrale, intră în funcţiune un microprogram realizat prin hardware, care încarcă în memoria principală baiţii 0—23 de pe discul rezident. În faza a doua, cei 24 baiţi încărcaţi ajută la încărcarea integrală a programului *IPL*. Prima şi a doua fază durează aproximativ o secundă. Urmează un mesaj la consolă prin care *IPL* cere să se comunice data zilei. După comunicarea datei zilei, unitatea centrală emite un alt mesaj la consolă prin care anunţă că *IPL* este încărcat complet. În continuare, programul *IPL* încarcă programul supervizor şi aduce la zero toată memoria neocupată de acesta. După aceea sistemul de calcul rămîne în starea de aşteptare. Scoaterea sistemului din această stare se face prin apăsarea a două taste de pe consolă. Prin aceasta, *IPL* încarcă în *BG* programul Job Control ca să pregătească pentru execuţie primul program al utilizatorului, cu care funcţiunile sale s-au terminat.

Programul Job Control este destinat pregătirii sistemului de calcul în vederea executării unui program. Am văzut că, pentru primul program, *JC* este încărcat de *IPL*. Pentru fiecare din următoarele programe de executat, *JC* este încărcat de supervizor la terminarea fiecărui program. După terminarea activităţilor sale, programul *JC* se şterge din *BG*. O funcţie importantă a programului Job Control este aceea de asignare a unităţilor fizice la nume simbolice şi de prelucrare a informaţiei privind etichetele fişierelor de pe discuri şi benzi (vezi cap. IV).

Programul Linkage Editor este un program de deservire a sistemului. După translatarea programelor sursă, programele obiect obţinute au adrese relative şi deci nu pot fi încărcate în memoria principală şi executate. Programul *LE* alocă adrese absolute şi aduce programele obiect în stare executabilă. Această operaţie complexă se numeşte *editare*. El editează programele unul cîte unul, în ordinea dorită de utilizator. După ce programul a fost editat, programul în stare executabilă este depus pe discul rezident în *CIL*, adică este catalogat. Programul catalogat temporar este încărcat de subprogramul de încărcare din supervizor în memorie şi executat, după care el se şterge din *CIL*.

Scanned with OKEN Scanner

Programul bibliotecar (*Librarian*) este format din mai multe programe cu funcţiuni independente. Deci ne putem referi la acest program şi prin denumirea de programe de deservire a bibliotecilor sistemului.

Programul bibliotecar are trei funcţiuni de bază : întreţinere, afişare şi copiere. Funcţia de întreţinere a bibliotecilor o are programul *MAINT* care poate executa următoarele operaţii individuale :

— cataloghează programe în *RL* şi *SSL* ;
— şterge programe din *CIL*, *RL* şi *SSL* ;
— şterge tot conţinutul bibliotecilor *RL* sau *SSL* ;
— condensează conţinutul celor trei biblioteci.

Prin afişare se înţelege expunerea pe un anumit mediu a unui conţinut. Funcţia de afişare a conţinutului bibliotecilor este executată de programele *CSERV* pentru *CIL*, *RSERV* pentru *RL* şi *SSERV* pentru *SSL*. Afişarea se face la unitatea asignată la *SYSLST*.

Funcţia de copiere a conţinutului bibliotecilor este executată de programul *CORGZ*. Copierea poate fi parţială sau integrală şi se face prin înscrierea conţinutului copiat pe un alt disc. Programul *CORGZ* mai poate fi folosit la crearea şi copierea bibliotecilor private de tip *RL* şi *SSL*.

Programele Supervizor, Job Control, Linkage Editor, programele de deservire a bibliotecilor şi programele de translatare formează conţinutul minimal al bibliotecii *CIL*.

1.5.5. Prelucrarea independentă de timp, prelucrarea în timp real, prelucrarea prin divizarea timpului, multiprogramrea şi multiprelucrarea

Prin prelucrare se înţelege orice fel de prelucrare a informaţiei într-un sistem de calcul. Ne vom ocupa de cîteva modalităţi de prelucrare care sînt întîlnite în prezent foarte des. Fiecare modalitate presupune existenţa unui sistem de operare adecvat şi a unei configuraţii de calcul care să suporte prelucrarea respectivă.

Prelucrarea independentă de timp asigură executarea programelor unul după altul, fiecare program fiind depus succesiv în aceeaşi partiţie a memoriei. Deci memoria este împărţită doar în domeniul sistemului de operare şi domeniul de prelucrare a programului. Executarea unui program este independentă de timp, în sensul că unitatea centrală execută programul pînă la terminarea lui, după care începe executarea unui nou program. În timpul prelucrării, are loc simultaneitatea executării operaţiilor de calcul în unitatea centrală şi operaţiilor de in-

Scanned with OKEN Scanner

trare/ieşire, realizată prin existenţa canalelor. Dispozitivele de intrare/ieşire sînt distribuite în fiecare moment unui singur program.

Prelucrarea în timp real introduce dependenţa de timp a executării unui program. Ea se întîlneşte la sistemele de calcul care controlează desfăşurarea unor procese într-o uzină, într-un combinat, etc. La prelucrarea în timp real, desfăşurarea procesului cu care este legat sistemul este funcţie de timpul în care procesul primeşte rezultatul prelucrării informaţiei, adică sistemul de calcul nu poate să întîrzie cu livrarea rezultatelor peste un anume timp critic. Programul care prelucrează informaţia se află permanent în memoria principală. Datele pot să sosească la intervale de timp fixe sau variabile. Începerea prelucrării datelor este realizată de ceasul unităţii centrale, iar obţinerea rezultatelor se face la un moment sau un interval de timp bine precizat. De obicei, sistemele care prelucrează informaţia în timp real au posibilitatea să execute şi programe independent de timp, atunci cînd nu există informaţie de prelucrat de la proces. Dar, în momentul apariţiei acestei informaţii, se întrerupe executarea programului independent de timp, cu memorarea conţinutului registrelor de bază şi se execută programul ce prelucrează informaţia de la proces. Acest program are prioritate absolută. Programul care a fost întrerupt este continuat după ce se termină prelucrarea informaţiei de la proces. O astfel de prelucrare există, de exemplu, la calculatorul Elliott 803 la Combinatul Siderurgic din Hunedoara.

Prelucrarea prin divizarea timpului este o modalitate de a evita blocarea unităţii centrale şi a echipamentelor periferice de un singur utilizator pe toată durata executării programului său. Acest tip de prelucrare se întîlneşte de obicei cînd la o unitate centrală puternică sînt conectate prin linii de telecomunicaţii mai mulţi utilizatori, fiecare utilizator avînd dispozitivele sale de intrare/ieşire, legate direct cu sistemul de calcul. Fiecărui utilizator i se acordă succesiv o anumită cantitate de timp — de exemplu, tranşe de cîte 100 ms — cînd poate folosi sistemul de calcul. După expirarea tranşei de timp acordate, se memorează registrele de bază pentru reluarea ulterioară a programului întrerupt şi se acordă timp altui utilizator. Comutarea controlului de la un utilizator la altul se face automat prin acţiunea ceasului unităţii centrale şi a sistemului de operare.

Utilizatorii au priorităţi determinate la acordarea tranşelor de timp (de lungimi egale). Această distribuire a timpului pe bază de planificare este cunoscută sub numele de *time sharing*. Pe durata din tranşa acordată, în care programul unui utilizator nu foloseşte unitatea centrală, fiind ocupat cu operaţii de intrare/ieşire, sistemul de operare permite utilizatorului cu prioritatea următoare să folosească acest timp pentru programul său. Priorităţile se succed circular, adică 1,

Scanned with OKEN Scanner

2, ..., n, 1,2, ... În figura 11 se dă un exemplu de distribuire planificată a timpului pentru trei utilizatori U_1, U_2, U_3, cu prioritățile 1, 2, 3. Fiecăruia i se acordă timpul T.

Timpul în care sistemul de operare decide cărui utilizator trebuie să i se dea controlul este foarte mic și deci el, practic, nu afectează perioada T.

Multiprogramarea este o formă de prelucrare a informației prin care la un moment dat, într-un sistem de calcul, există simultan mai multe programe active în diferite faze de execuție. Spre deosebire de prelucrarea cu divizarea timpului, cînd utilizatorul își introduce programul

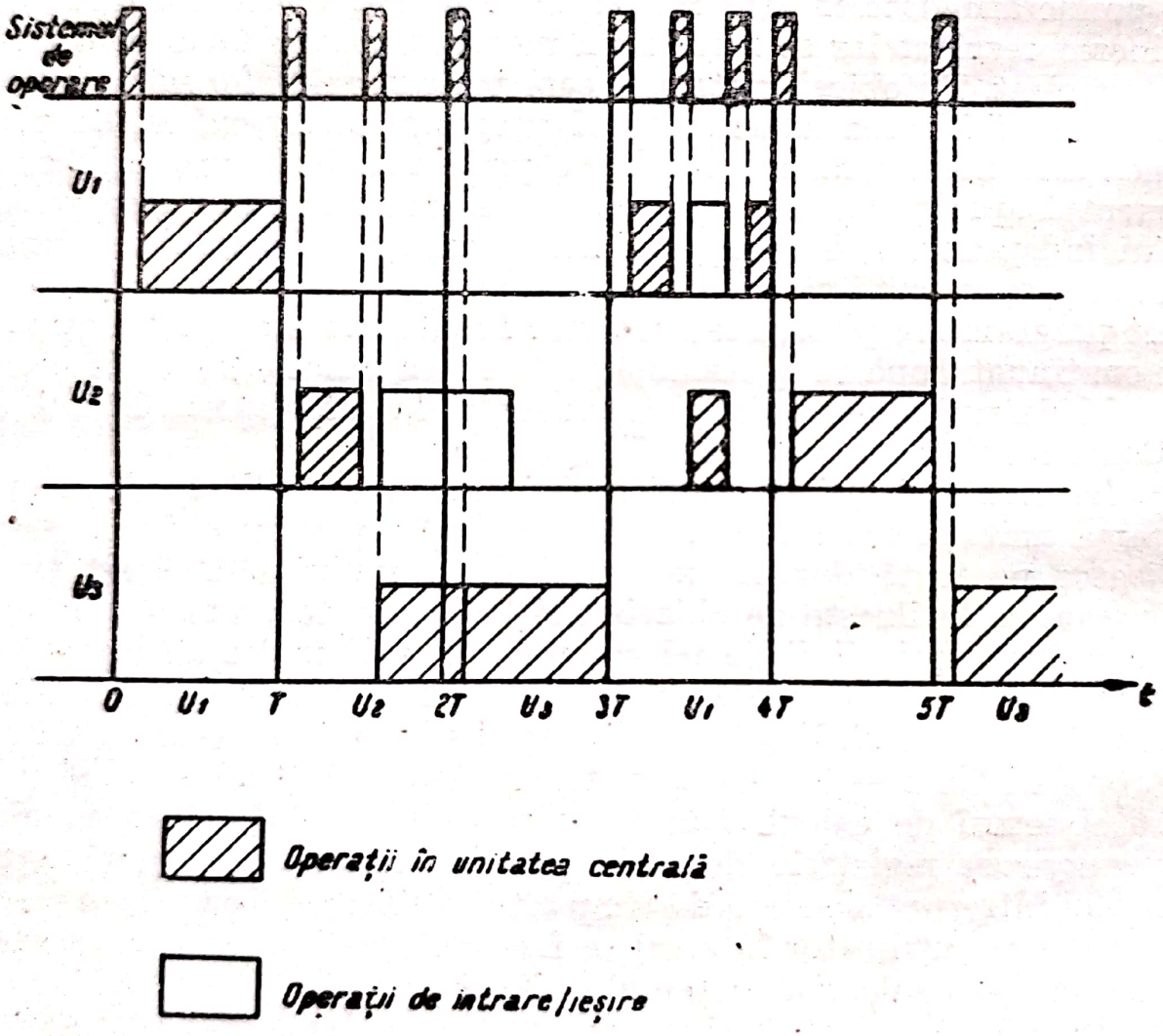


Fig. 11. Schemă de distribuire a timpului

atunci cînd i se acordă prima tranșă T de timp, la multiprogramare, programele de executat se introduc de la început în memoria principală. Pentru aceasta, memoria principală este împărțită într-un număr N de zone (domenii) protejate prin chei de protecție. Deci, în fiecare

Scanned with OKEN Scanner

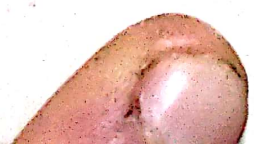
moment pot exista în memorie cel mult N programe. Priorităţile programelor pentru execuţie coincid cu priorităţile zonelor în care se găsesc. Astfel, programul din prima zonă va avea cea mai mare prioritate. Mărimile zonelor în care este împărţită memoria sînt la dispoziţia operatorului sistemului, care încarcă programele. Pentru fiecare program se asigură unităţile de intrare/ieşire necesare.

Vom da ca exemplu de realizare a multiprogramării, folosirea domeniilor F_1, F_2, BG care au priorităţile 1,2 şi respectiv 3. Deci pot exista simultan trei programe în memorie. La început controlul este dat domeniului F_1. Programul din F_1 este executat pînă cînd solicită un transfer de date (de la intrare sau spre ieşire). Programul supervizor transmite cererea de intrare/ieşire canalului, iar F_1 trece în stare de aşteptare, fapte consemnate într-o listă de stare. Controlul este comutat lui F_2. Executarea programului din F_2 are loc pînă cînd F_1 a terminat cu transferul datelor sau pînă cînd F_2 solicită o operaţie de intrare/ieşire. Cînd F_1 a terminat transferul, F_2 se întrerupe şi se comută controlul lui F_1. Dacă F_1 nu a terminat transferul, dar F_2 solicită o operaţie de intrare/ieşire, faptul acesta este consemnat în lista de stare, F_2 se întrerupe, iar controlul este comutat lui BG. Supervizorul urmăreşte tot timpul lista de stare şi comută controlul la programul cu prioritate superioară, de îndată ce acesta a terminat transferul de date. În concluzie, în orice moment, numai unul din programe este executat, celelalte fiind inactive pentru că îşi satisfac operaţii de intrare/ieşire sau sînt în stare de aşteptare, avînd prioritatea mai mică decît a programului activ. Totdeauna controlul este comutat programului în aşteptare, cu prioritatea cea mai mare.

Dacă unitatea centrală dispune de o singură consolă de comunicare cu operatorul, pentru toate zonele în care este împărţită memoria, spre a şti din partea cărui program a apărut mesajul tipărit la consolă, el are în faţă indicatorul F_1, F_2 sau BG. Răspunsul şi acţiunile operatorului trebuie să fie corespunzătoare cu mesajele.

Multiprogramarea implică o prelungire (mică) a duratei de execuţie a fiecărui program, dar randamentul sistemului într-un interval de timp este mai mare decît la prelucrarea independentă de timp.

Multiprelucrarea presupune fie existenţa unui calculator central şi a unui număr de calculatoare satelit, fie existenţa unei singure unităţi centrale, cu o singură unitate de comandă, dar cu mai multe memorii principale şi mai multe dispozitive aritmetice-logice.



Scanned with OKEN Scanner

1.5.6. Despre generarea sistemului de operare al unei configurații de calcul

Activitățile unei configurații moderne de calcul sînt controlate și dirijate de sistemul de operare. Acesta este depus pe un disc sau pe o bandă și formează rezidența sistemului. O familie de calculatoare poate avea mai multe tipuri de sisteme de operare. De exemplu, pentru IBM 360 am văzut că există *BPS, TOS, BOS, DOS, OS, TSS.*

Presupunem că avem o configurație de calcul recent montată, care urmează să înceapă să funcționeze. Întîi trebuie ales tipul sistemului de operare. Alegerea este funcție de: 1. posibilitățile fizice ale configurației — de exemplu, configurația care nu dispune de discuri nu va putea folosi un sistem de operare care necesită discuri; 2. posibilitățile unității centrale și în primul rînd de capacitatea memoriei — de exemplu, pentru seria IBM 360, o unitate centală cu memoria 32 *K* nu va putea folosi *OS*; 3. dorințele utilizatorilor în exploatarea configurației. După ce tipul a fost stabilit, urmează crearea efectivă a sistemului de operare pentru configurația respectivă, adică generarea sistemului. În mod concret ne vom referi la generarea sistemului de operare pentru o configurație cu discuri magnetice. Se pornește de la un *sistem de operare minimal*, de tipul celui ales, trimis sau cumpărat de la firma constructoare a sistemului de calcul. El se află pe un disc sau pe o bandă și ajută la crearea și implementarea sistemului de operare al configurației. Dacă configurația dispune de discuri, rezidența se creează de obicei pe un disc. Sistemul de operare minimal conține cel puțin programele de control ale sistemului de operare. Acest lucru este absolut necesar pentru că, pentru generare, sistemul minimal prelucrează un program de generare întocmit de cel care generează sistemul. Programul conține informații despre configurația concretă de calcul și despre dorințele de exploatare ale utilizatorilor. Aceste dorințe sînt exprimate numai la generare, după care toți utilizatorii (noi) sînt obligați să se conformeze.

Programul de generare trebuie să răspundă la un număr destul de mare de întrebări puse de sistemul minimal, pentru a putea crea sistemul configurației. De exemplu, el specifică dacă se folosesc sau nu: unitățile de discuri, multiprogramarea, multiprelucrarea, protecția memoriei, virgula mobilă, ceasul unității centrale, etc. Specifică apoi date despre configurația de calcul: numărul și tipul canalelor, numărul și tipul unităților de discuri, de benzi, de cititoare de cartele sau de bandă de hîrtie, de imprimante, etc. Tot programul de generare mai indică zonele în care este împărțită memoria principală, numărul maxim de nume simbolice ale utilizatorului și cîte din acestea sînt dis-

Scanned with OKEN Scanner

tribuite pentru fiecare zonă din partiție. Se indică apoi asignările standard și adresele perifericelor din cadrul configurației. O parte importantă a programului de generare este aceea prin care se indică bibliotecile sistemului de operare și dimensiunile lor pe discul care formează rezidența. Pentru fiecare bibliotecă în parte se cataloghează permanent un număr de programe, după dorința celui care realizează generarea. Bibliotecile absolute necesare pentru sistemul de calcul generat — de obicei, una, de exemplu biblioteca *CIL* — trebuie să aibă cel puțin un conținut minimal format din programul supervizor, programul Job Control, programul de editare a legăturilor și de alocare a adreselor absolute, programe de translatare și de întreținere a bibliotecilor. Una din instrucțiunile programului de generare indică numărul de adrese ocupat de programul supervizor în memoria principală, cînd acesta este încărcat de pe rezidență. De la această adresă începe domeniul de prelucrare al programelor utilizatorilor.

După ce sistemul de operare a fost generat, adică s-a creat pe un disc rezidența sistemului, sistemul de operare minimal nu mai este folosit. Cînd se începe lucrul la calculator, rezidența se montează pe o unitate de discuri. Adresa acestei unități de discuri se anunță unității centrale prin poziționarea a trei comutatoare circulare de pe panoul unității centrale. Primul comutator indică numărul canalului, al doilea indică numărul unității de comandă a discului și al treilea indică numărul unității de discuri. Referindu-ne la IBM 360/30 descris, prin aceasta unitatea centrală știe de unde să încarce programul *IPL*, iar mai departe acesta știe de unde să încarce programul supervizor.

De generarea corectă și de catalogarea în bibliotecile sistemului a unor programe cît mai judicios alese este legată desfășurarea întregii activități a configurației de calcul.

1.6. FOLOSIREA BIBLIOTECILOR DE SUBPROGRAME

1.6.1. Ce este o bibliotecă de subprograme a unui centru de calcul

Există probleme care necesită executarea de mai multe ori a aceluiași tip de calcule, dar cu date diferite. Un program scris de utilizator într-un anumit limbaj de programare trebuie să asigure executarea repetată a grupului de calcule necesar în problema respectivă. Aceasta se poate face prin includerea în programul scris de utilizator a grupului de calcule ce se repetă de atîtea ori cît necesită problema de rezolvat. Dar în acest mod se lungește foarte mult programul și se mărește

Scanned with OKEN Scanner

timpul de compilare al acestuia. Înlăturarea acestor neajunsuri se face prin constituirea grupului de calcule ce se repetă ca o secţiune separată a programului, numită *subprogram* sau *subrutină*. Programul propriu-zis devine *program principal*. El face apel la subprogram ori de cîte ori are nevoie. Chemarea subprogramului se face cu ajutorul numelui pe care acesta îl are. Numele este dat de cel care a construit subprogramul, după dorinţa sa. În afară de nume, pentru subprogram se mai indică un număr de parametri. Aceşti parametri sînt parametri fictivi, în sensul că ei sînt doar menţionaţi, fără a avea valori atribuite. Ei trebuie să aibă valori atribuite doar în momentul în care programul principal face apel la subprogram.

Fie de exemplu subprogramul *INVMAT* de inversare a unei matrice. Matricea care se inversează este *A* şi are *N* linii şi *N* coloane. Presupunem că matricea inversată (tot cu *N* linii şi *N* coloane) este notată *B*. Atunci parametrii sînt *A*, *B*, *N*. *N* poate avea orice valoare întreagă care conduce la dimensiuni ale lui *A* şi *B* pe care le admite memoria principală. În momentul apelării lui *INVMAT*, parametrii trebuie să aibă nume şi valori determinate, de exemplu *INVMAT* (*MATR*1, *MATR*2, 10). Aceasta înseamnă că, folosind subprogramul *INVMAT*, se inversează matricea pătratică cu 10 linii numită *MATR*1 şi se obţine matricea inversă *MATR*2.

Subprogramele sînt de obicei în aşa fel construite încît ele nu tipăresc rezultatele calculelor pe care le execută. Instrucţiunile de tipărire sau de extragere a acestor rezultate din memorie se dau prin programul principal. Dar pentru ca rezultatul să se păstreze în memorie pînă la tipărirea lui, este necesar ca acel cîmp din memorie care îl păstrează să fie declarat ca parametru fictiv al subprogramului. În exemplul anterior, cîmpul în care se obţine rezultatul este *B* (căruia îi corespunde *MATR*2). Obţinerea matricei inverse se face prin cîteva instrucţiuni de tipărire a conţinutului lui *MATR*2.

Un alt mare avantaj al subprogramelor este acela că ele pot fi catalogate o singură dată şi apelate de cîte ori este nevoie. Scrierea lor nu trebuie să fie făcută de cel care le utilizează. Deci un subprogram poate să fie folosit de orice utilizator. Condiţia folosirii lui este aceea de a-i cunoaşte numele, parametrii şi unde a fost catalogat. Catalogarea se poate face în bibliotecile sistemului sau într-o bibliotecă privată. Dacă a fost catalogat într-o bibliotecă privată, trebuie să se ştie dacă ea se afla pe un disc sau pe o bandă. Apoi trebuie să se cunoască numărul discului sau numărul rolei de bandă care conţine subprogramul dorit. Pentru activarea bibliotecii private, trebuie să se facă corespondenţa dintre adresa perifericului unde este montată biblioteca şi numele simbolic al bibliotecii, adică trebuie făcută asignarea.

Scanned with OKEN Scanner

Prin apelarea unui subprogram dintr-o bibliotecă, el nu se șterge din bibliotecă. Totalitatea subprogramelor care există la un centru de calcul și care pot fi folosite de utilizatori formează *biblioteca de subprograme a centrului de calcul*. Rezolvarea diverselor tipuri de probleme este cu atît mai ușoară cu cît biblioteca de subprograme este mai bogată.

În general, fiecare subprogram din bibliotecă are indicații scrise asupra posibilităților subprogramului, asupra algoritmului pe care îl folosește, asupra numelui și parametrilor săi și asupra modului de folosire. Pe lîngă aceasta, dacă subprogramul se află într-o bibliotecă privată sau dacă este de sine stătător pe un disc sau pe o bandă, trebuie să existe indicații scrise asupra activării acestora. Cu alte cuvinte, putem spune că fiecare subprogram este însoțit de o documentație. Studierea de către utilizator a acestei documentații este absolut obligatorie pentru folosirea optimă a subprogramului.

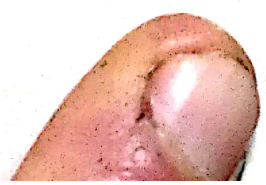
1.6.2. Descrierea folosirii unor subprograme dintr-o bibliotecă concretă

Ne vom referi la cîteva exemple de subprograme care pot fi folosite la configurația descrisă în § 1.2.6. Începem prin a prezenta biblioteca privată a subprogramelor cu aplicații la cercetarea științifică, numită pe scurt *SSP* (Scientific Subroutine Package). Biblioteca este de tip relocatabil (*RL*) și este depusă pe pachetul de discuri nr. 27 (la *CCUB*). Conținutul ei se referă la analiza numerică și la statistica matematică. Relativ la analiza numerică, biblioteca conține subprograme pentru:

— rezolvarea sistemelor de ecuații liniare;
— algebră matriceală (adunări, înmulțiri, inversări de matrice, etc.);
— vectori și valori proprii;
— diferențiere și integrare numerică;
— rezolvarea ecuațiilor diferențiale;
— rezolvarea sistemelor de ecuații diferențiale;
— serii Fourier;
— funcții Bessel, funcții Gamma;
— rezolvarea ecuațiilor algebrice de ordinul n;
— rezolvarea ecuațiilor transcendente;
— polinoame Cebîșev, Hermite, Legendre;
— aproximare și interpolare, etc.

Relativ la statistica matematică biblioteca conține subprograme pentru:

— generare de numere aleatoare;



Scanned with OKEN Scanner

— funcţii de repartiţie ;
— teste neparametrice ;
— calculul regresiei (regresia liniară multiplă, analiza în paşi a regresiei — stepwise regression) ; .
— analiza corelaţiilor unui număr de variabile (factor analysis).

În continuare vom explica cum se foloseşte un subprogram din *SSP* într-un program scris de utilizator în Basic Fortran IV şi în Fortran IV. Avînd în vedere că subprogramele din *SSP* nu conţin instrucţiuni de tipărire a rezultatelor, programul principal (calling program) trebuie să conţină cel puţin aceste instrucţiuni pe lîngă instrucţiunea de chemare a subprogramului. Tipărirea se face atunci conform dorinţelor utilizatorului.

Chemarea unui subprogram se poate face numai după ce biblioteca *SSP* a fost activată. Pentru aceasta se montează pachetul nr. 27 pe o unitate de discuri. Apoi se anunţă sistemului de operare adresa acestei unităţi printr-o instrucţiune de asignare. Numele simbolic al bibliotecii private este *SYSRLB*. Asignarea se face direct la consolă sau printr-o cartelă. Instrucţiunea pe cartelă începe din coloana 1 şi are următoarea formă (dacă presupunem că folosim unitatea 193) :

//bASSGNbSYSRLB, X'193'

Prin *b* am notat spaţiul liber (blank).

Translatarea în program obiect a subprogramelor sursă *SSP* s-a făcut cu translatorul Basic Fortran IV. Dacă programul principal este scris de utilizator în Fortran IV, atunci trebuie activat, din sistemul de operare, un subprogram de interpretare a instrucţiunilor din Fortran IV în Basic Fortran IV. Aceasta se face prin următoarea instrucţiune perforată pe cartela din coloana 2 :

INCLUDE ILFACOM

Tabelul 7 indică ordinea şi conţinutul cartelelor care trebuie să preceadă programul principal scris în Basic Fortran IV sau în Fortran IV.

Pentru semnificaţia cartelelor *JOB*, *OPTION* şi *EXEC* se pot consulta [1] şi [2]. Pentru ca utilizatorul să se asigure că nu va începe executarea programului său fără ca pachetul 27 să fie montat (cînd programul ar fi anulat), se atrage atenţia operatorului printr-o cartelă cu următorul conţinut (din coloana 1).

//bPAUSE VA ROG MONTATI DISCUL 27 PE ———

Această cartelă este plasată înaintea cartelei de asignare. Imediat după ce această cartelă a fost citită prin cititor, unitatea centrală

Scanned with OKEN Scanner

TABELUL 7

Conținutul cartelelor	
Basic Fortran IV	Fortran IV
// JOB	// JOB . . . . .
// OPTION.	// OPTION. . . . .
// ASSGN SYSRLB, X'	// ASSGN SYSRLB, X' . . . .
// EXEC FORTRAN	bINCLUDE ILFACOM
	// EXEC FFORTRAN

intră în starea de așteptare, iar mesajul transmis de utilizator operatorului este tipărit la consolă. După ce discul 27 a fost montat, operatorul dă comanda de continuare a prelucrării programului. Chemarea propriu-zisă a subprogramului care ne interesează se face prin instrucțiunea *CALL* nume (parametrii), plasată în programul principal. Numele și parametrii se iau conform descrierii acestora în [3]. Exemple de folosire a bibliotecii *SSP* se găsesc în cap. V.

Subprogramele de bibliotecă pot fi și de sine stătătoare, adică neincluse într-un grup mare, ca în cazul bibliotecii private *SSP*. Un astfel de subprogram este subprogramul de determinare a optimului unei funcții liniare pe o mulțime de restricții linare (vezi cap. II). Subprogramul rezolvă deci o problemă de programare lineară. Ne vom referi la el prin *LPC* (Liinear Programming Code).

Subprogramul *LPC* se află depus pe banda 100 (la *CCUB*). Pentru folosirea lui, el trebuie catalogat (permanent) în *CIL*. Pentru catalogare, este suficient să executăm următoarele operații:

1. montăm banda 100 pe unitatea de bandă cu adresa 180 sau 181;
2. dezasignăm cititorul de cartele prin comanda dată la consolă

ASSGN SYSIN,UA

3. asignăm *SYSIN* la 180 (sau 181) prin comanda dată la consolă:

ASSGN SYSIN, X'———'

(Cum în mod obișnuit utilizatorul nu se află în sala calculatorului cînd i se execută programul, toate aceste indicații i se pot da operatorului în scris). Programul catalogat în *CIL* are numele *PLINROOT* și deci activarea lui se va face prin *//bEXEC PLINROOT*. După acea-

stă cartelă, urmează cartele care dau valori parametrilor pe care îi cere subprogramul, iar după aceste cartele urmează cartelele de date ale problemei. În cap. V se găseşte un exemplu complet de folosire a subprogramului *LPC*. Descrierea tuturor posibilităţilor de folosire ale acestui subprogram se află în [4], [8] şi [9].

BIBLIOGRAFIE

[1] IBM System/360 — *FORTRAN IV Language.* C28—6515—7

[2] IBM System/360 *DOS* — *FORTRAN IV, Programmer's Guide,* C28—6397—0

[3] IBM System/360 (360*A*—*CM*—03*X*) — *Scientific Subroutine Package, Programmer's Manual,* Version III. H20—0205—3

[4] IBM Belgium — *Linear Programming Code, DOS*/360. 15.2.701

[5] Klerer, M., Korn, G. — *Digital Computer User's Handbook,* McGrow Hill, New York, 1967.

[6] Kitov, A. I., Krinițki, N. A. — *Maşini electronice cifrice şi programare,* Ed. Tehnică, Bucureşti, 1963.

[7] Ledley, R.S. — *Programarea şi utilizarea calculatoarelor numerice,* Ed. Tehnică, Bucureşti, 1968.

[8] Linear Programming System/360 — *Program Description Manual,* 360*A*—*CO*—18*X*. H20—0607—0

[9] Mathematical Programming System/360 — *Linear and separable Programming, User's Manual,* Version II. 360 *A*—*CO*—14*X*. H20—0476—2

[10] Pélegrin, M. — *Calculatoare analogice şi numerice* Ed. Tehnică, Bucureşti, 1966.

[11] Sippl, Ch. — *Computer Dictionary and Handbook* Howard Bobbs-Merill, New York, 1967.

Scanned with OKEN Scanner

Capitolul II

MODELE MATEMATICE ŞI ALGORITME PENTRU PROBLEME DE CERCETARE ŞTIINŢIFICĂ

2.1. MODELUL MATEMATIC ŞI ALGORITMUL DE REZOLVARE AL UNEI PROBLEME. IDEEA DE CALCULABILITATE

2.1.1. Modele şi algoritme

Rezolvarea diverselor probleme face parte integrantă din activitatea zilnică umană. În cele ce urmează ne vom ocupa numai de problemele care au un aspect calculatoriu, definite *direct* prin relaţii matematice sau definite *descriptiv* prin propoziţii şi fraze. La definirea descriptivă a problemei, exprimarea este vagă şi uneori neclară. De exemplu: să se studieze posibilitatea îmbunătăţirii transportului cimentului de la unităţile producătoare la unităţile consumatoare. Exprimarea este vagă, dar ideea este clară: îmbunătăţirea transportului cimentului. Rămîne să precizăm condiţiile detaliate ale problemei: care sînt unităţile producătoare, care sînt unităţile consumatoare, cu ce mijloace şi pe ce rute se efectuează transportul, care sînt distanţele dintre unităţi, care este costul transportului (tonă/km), etc. Problema definită descriptiv, care răspunde la toate aceste întrebări, este o problemă *complet definită*. Trebuie cercetat dacă ea este şi *corect definită*, de exemplu dacă suma cererilor nu depăşeşte suma cantităţilor produse.

Din cele spuse mai sus deducem că primul pas în rezolvarea unei probleme este definirea corectă şi completă a ei. (Vom numi tot problemă şi o parte de sine stătătoare a unei probleme laborioase). Urmează o etapă importantă şi anume *etapa de analiză a problemei*. Analizarea problemei constă în precizarea şi cunoaşterea legilor şi a fenomenelor care o guvernează. Analiza teoretică a problemei este punctul de plecare în elaborarea unui *model*. Activitatea de definire a unui model se numeşte *modelare*. După cum se arată în [5], modelarea nu este o activitate nouă. Ea a fost folosită de la începutul istoriei umane pentru reprezentarea obiectelor sau a ideilor. De exemplu, un model de aparat de zbor este avionul. Ştiinţa se bazează fundamental pe noţiunea de modelare, iar progresul ştiinţei este progresul modelării fenomenelor naturale şi a ideilor.

Scanned with OKEN Scanner

Dar, de fapt, ce este acela un model? Un răspuns poate fi acesta: modelul este o reprezentare a ceea ce este esenţial şi total într-o problemă sau într-un fenomen. Prin aceasta am exprimat şi *scopul* construirii modelului. În [5] se spune că elaborarea unui model este o artă, care impune echilibrarea artistică a unei unităţi de contrarii. Într-adevăr, un model trebuie să fie o abstractizare a problemei sau a fenomenului pe care le reprezintă, dar în acelaşi timp trebuie să fie o reprezentare cît mai fidelă a acestora. Realizarea acestui echilibru este o sarcină dificilă şi foarte importantă.

Folosirea modelelor în rezolvarea problemelor este o necesitate obiectivă. În decursul timpului s-au acumulat cunoştinţe despre rezolvarea unui anumit tip de problemă. Toate problemele de acel tip au fost abstractizate, adică au fost reprezentate printr-un model. De exemplu, modelul unei probleme de transport este valabil pentru transportul cimentului, al cărbunelui, distribuirea forţei de muncă, transportul ideilor, etc. Pentru rezolvarea problemei transporturilor există metode matematice bine studiate care asigură obţinerea soluţiei optime. Încadrarea unei probleme concrete în modelul problemelor de transport se poate face direct, fără nici o modificare a conţinutului problemei sau se poate face numai cu preţul renunţării la unele elemente din definirea acesteia. Dacă elementele la care se renunţă nu afectează esenţial conţinutul problemei, atunci modelul este gata construit şi metodele matematice sînt cunoscute. Altfel, se caută încadrarea problemei concrete în alt tip de problemă, pentru care există model şi metode matematice de rezolvare şi care nu afectează sau afectează, în condiţii acceptabile, conţinutul problemei. De multe ori însă, este necesară construirea unui model nou şi găsirea mijloacelor matematice adecvate de rezolvare a problemei.

Există multe criterii de clasificare a modelelor. O primă clasificare dată în [3] împarte modelele în trei tipuri fundamentale: *iconice* (de la icon, care însemnează icoană, imagine, statuie), *fizice-analogice* şi *matematice* sau *simbolice*. Un model iconic este o reprezentare la o scară redusă sau la o scară mărită a unui anumit obiect fizic, de exemplu, modelul unui tip nou de automobil sau modelul unui atom. Modelul fizic-analogic se bazează pe substituirea unei proprietăţi sau a unei caracteristici reale printr-o proprietate sau o caracteristică intermediară. De exemplu, pentru a se studia comportarea la comprimare şi destindere a unui arc elicoidal, se studiază cu ajutorul unui calculator analogic variaţia tensiunii electrice. Deci, la început problema este rezolvată în forma substituientă şi apoi soluţia obţinută se proiectează asupra originalului. Modelul matematic este o reprezentare abstractă, realizată cu ajutorul scrierii simbolice, a problemei care se cere rezolvată. Reprezentarea este funcţie de elementele pe care vrea

Scanned with OKEN Scanner

să le ia în considerare cel care face analiza problemei în scopul obținerii modelului. Din această frază deducem că pentru o problemă se pot realiza mai multe modele. Elementele unui model matematic sînt: variabilele, parametrii, relațiile funcționale și caracteristicile operative. Dacă elementele sistemului real ce se cere modelat matematic prezintă o variabilitate în evoluția sistemului, atunci acestor elemente li se asociază *variabile* numerice, logice sau de altă natură. Dimpotrivă, elementelor care prezintă stabilitate, sau constanță, în evoluția sistemului li se sociază *parametri*, care iau valori bine determinate. *Relațiile funcționale* reprezintă dependența dintre diversele variabile ale modelului, corespunzător cu dependența unora de altele a elementelor sistemului real. Deoarece uneori această dependență este greu de transpus în forma unor ecuații sau relații bine precizate, cel ce construiește modelul face anumite ipoteze asupra acestor relații sau asupra variabilelor modelului (în special cînd aceste variabile sînt aleatoare). Aceste ipoteze sînt cunoscute sub numele de *caracteristici operative*.

Unele variabile, sau parametri, pot fi presupuse *cunoscute* în sensul că se cunoaște variația lor sau se cunosc valorile lor particulare. Relațiile funcționale definesc o legătură între aceste elemente și celelalte variabile și parametri, care trebuie *determinate* prin rezolvarea sistemului de ecuații definit de aceste relații. Se pune deci problema alegerii caracteristicilor operative astfel încît să se poată obține elementele (variabile și parametri) necunoscute ale modelului, în funcție de elementele cunoscute.

Exemplul 1. Pentru exemplificare, vom prezenta pe scurt un model matematic al endemiei tuberculozei [12].

După cîte se știe, tuberculoza este o boală endemică (cu evoluție lentă), care are următoarele particularități:

a) Este o boală infecțioasă, cauzată de trecerea bacilului de la un individ la altul.

b) Un individ infectat nu este întotdeauna bolnav; bolnavii provin numai din persoane care au fost în prealabil infectate o perioadă îndelungată de timp (minimum un an).

c) Infecția este transmisă numai prin intermediul bolnavilor (care posedă focare emițătoare de bacili).

d) Vindecații rămîn totuși infectați.

e) O parte din infectați *pot* deveni neinfectați după o perioadă de timp.

f) Noii născuți sînt întotdeauna neinfectați. Deci populația unui teritoriu închis (fără migrații din sau în exterior, ca de exemplu un teritoriu național) se împarte în mai multe clase și anume: neinfectați, infectați, bolnavi. Ne interesează cum evoluează aceste clase în timp.

Scanned with OKEN Scanner

Modelul conține următoarele variabile:

t = timpul (variabilă de intrare, a cărei variație este cunoscută); timpul se măsoară în ani;

$P(t)$ = populația totală la momentul t;

$N(t)$ = populația de neinfectați la momentul t;

$I(t)$ = populația de infectați la momentul t;

$B(t)$ = populația de bolnavi la momentul t;

$I(t, t+1)$ = numărul de noi infectați în intervalul $(t, t+1)$;

$B(t, t+1)$ = numărul de bolnavi noi în intervalul $(t, t+1)$;

$n(t, t+1)$ = numărul de noi născuți în intervalul $(t, t+1)$;

$D(t, t+1)$ = numărul total de decedați în intervalul de timp $(t, t+1)$;

$v(t, t+1)$ = numărul celor vindecați în intervalul $(t, t+1)$.

Se va nota de asemenea $DN(t, t+1)$, $DI(t, t+1)$, $DB(t, t+1)$ respectiv numărul decedaților din neinfectați, infectați și bolnavi în intervalul de timp $(t, t+1)$.

Relațiile funcționale ale modelului care rezultă din particularitățile a) — f) ale tuberculozei sînt

(1) $I(t+1) = (1-\alpha)I(t) + I(t, t+1) + v(t, t+1) - DI(t, t+1) - B(t, t+1)$;

(2) $B(t+1) = B(t) + B(t, t+1) - DB(t, t+1) - v(t, t+1)$;

(3) $N(t+1) = N(t) + n(t, t+1) + \alpha I(t) - DN(t, t+1) - I(t, t+1)$;

(4) $P(t+1) = P(t) + n(t, t+1) - D(t, t+1)$;

(5) $P(t) = I(t) + B(t) + N(t)$;

(6) $D(t, t+1) = DN(t, t+1) + DI(t, t+1) + DB(t, t+1)$.

Aici *parametrul* α reprezintă proporția de neinfectați proveniți din rîndul infectaților. Relațiile (5) și (6) reprezintă *identități* ale modelului.

Alte relații funcționale sînt:

(7) $I(t, t+1) = f_1(B(t))$;

(8) $B(t, t+1) = f_2(I(t))$;

(9) $v(t, t+1) = f_3(B(t))$;

(10) $n(t, t+1) = f_4(P(t))$;

(11) $DI(t, t+1) = f_5(I(t))$;

(12) $DB(t, t+1) = f_6(B(t))$;

(13) $DN(t, t+1) = f_7(N(t))$,

Scanned with OKEN Scanner

unde f_i sînt funcţii presupuse cunoscute; aceste relaţii rezultă din ipotezele a) — f).

Variabila t este o variabilă a cărei variaţie se presupune cunoscută (variabilă independentă). Se cere să se determine toate celelalte variabile în funcţie de t, rezolvînd sistemul (1) — (13). Funcţiile f_i, $1 \leqslant i \leqslant 7$, reprezintă caracteristicile operative ale modelului.

În afară de determinarea variabilelor menţionate, trebuie găsite şi *variabilele indici*.

$$(14) \quad F_1(t) = \frac{I(t)}{P(t)}, \; F_2(t) = \frac{B(t)}{P(t)}, \; F(t) = F_1(t) + F_2(t);$$

$$(15) \quad \varphi(t) = \frac{DB(t,\, t+1)}{D(t,\, t+1)} = \frac{f_6(B(t))}{f_5(I(t)) + f_6(B(t)) + f_7(N(t))}.$$

Se constată că este suficient să se găsească numai funcţiile $I(t)$, $B(t)$ şi $N(t)$, deoarece toate celalalte se exprimă cu ajutorul acestora. Se poate observa că dacă f_i, $1 \leqslant i \leqslant 7$, sînt de forma

$$(16) \quad f_i(Y(t)) = a_i Y(t) + b_i,$$

unde a_i şi b_i sînt parametri cunoscuţi, sistemul de ecuaţii (1) — (13) se poate reduce la un sistem de *ecuaţii cu diferenţe* de forma

$$(1^*) \quad I(t+1) = u_1 I(t) + v_1 B(t) + h_1;$$

$$(2^*) \quad B(t+1) = u_2 I(t) + v_2 B(t) + h_2;$$

$$(3^*) \quad N(t+1) = u_3 I(t) + v_3 B(t) + w N(t) + h_3,$$

unde coeficienţii u_i, v_i, h_i, $1 \leqslant i \leqslant 3$ şi w depind de a_j, b_j, $1 \leqslant j \leqslant 7$ şi α. Rezolvarea sistemului (1*) — (3*) se poate face fie analitic [12], fie prin metode iterative. Rezolvarea modelului prin metode iterative se poate face şi în condiţii mai puţin restrictive, cînd funcţiile de forma $X(t,\, t+1) = W(t)$ sînt cunoscute sub forma unor tabele de valori, ca funcţii de t.

Din acest model se vede cum caracteristicile operative ale modelului joacă un rol esenţial în rezolvarea acestuia.

În modelarea matematică, de obicei, se porneşte cu un model simplu care treptat se tot îmbogăţeşte cu elemente noi din definirea problemei. Există deci o relaţie foarte strînsă între trei elemente esenţiale: *ipoteza făcută*, *modelul construit* şi *soluţia obţinută*. În paragrafele următoare ale acestui capitol se vor vedea şi alte exemple de modele matematice.

Scanned with OKEN Scanner

Rezolvarea unui model matematic este legată de folosirea unui sistem ordonat de reguli de efectuare a calculelor, adică rezolvarea este legată de existența unui *algoritm*. Algoritmul este una din modalitățile de a pune în evidență ideea de calculabilitate.

2.1.2. Conceptul de calculabilitate

În [8] se spune că dezvoltarea matematicii a impus delimitarea acelei părți a ei, care poate fi introdusă într-o mașină (de calcul). În acest fel s-a născut matematica constructivă, a cărei noțiune de bază este aceea de *funcție calculabilă*. Funcția calculabilă este o funcție ale cărei valori pot fi obținute cu ajutorul unui *algoritm normal* [8], care este determinat printr-o schemă ordonată S de reguli.

O altă modalitate de definire riguroasă a conceptului de calculabilitate a fost imaginată de A. M. Turing prin modelul matematic de mașină de calculat, numită *mașină Turing*. Calculabilitatea în sensul algoritmului normal este echivalentă cu cea în sensul mașinii Turing. Între ele există numai deosebiri de ordin metodologic [8], care conferă grade diferite de comoditate în problemele atît de variate în care apare conceptul de calculabilitate. O mașină Turing are o unitate centrală care are un număr finit de stări, o memorie infinită și un dispozitiv de citire și de scriere. Pentru că memoria este infinită, mașina Turing este un calculator ideal. Cu toate acestea, funcții destul de simple nu pot fi calculate decît cu ajutorul unor mașini Turing cu stări interne foarte complicate. De aceea se utilizează o tehnică de compunere a unor mașini standardizate. Această tehnică permite ca ieșirile unei mașini Turing să devină intrările altei mașini Turing. De exemplu, pentru a calcula $E = (A + B) \cdot C$ se procedează așa: $D = A + B$ și deci $E = D \cdot C$.

A treia modalitate de definire a conceptului de calculabilitate, echivalentă cu celelalte două prezentate anterior, este cea cu ajutorul *funcțiilor recursive*, prin care calcularea valorilor unei funcții se reduce la aplicarea operațiilor aritmetice elementare.

Cele trei variante echivalente de definire a conceptului de calculabilitate nu sînt sigurele, dar sînt cele mai importante.

Scanned with OKEN Scanner

2.2. SCHEME LOGICE

2.2.1. Scheme logice generale

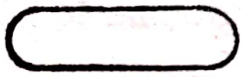
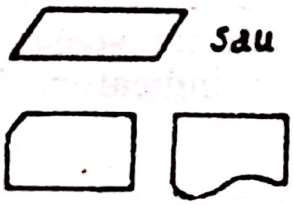
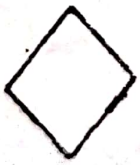
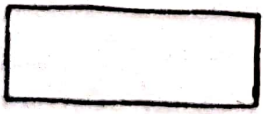
Am văzut că algoritmul normal este determinat de o schemă ordonată S, care indică succesiunea calculelor în rezolvarea unei probleme. Vom spune pe scurt că S este schema algoritmului. Pentru a înțelege și a urmări mai ușor o schemă de calcul furnizată de un algoritm, se folosește o reprezentare grafică a acesteia, numită *schemă logică*. După etapa de analiză a problemei, întocmirea schemei logice, adică punerea algoritmului sub formă grafică, este un principiu general în rezolvarea unei probleme. Pe baza schemei logice se va întocmi programul într-un limbaj algoritmic de programare, care urmează să fie executat de calculator. Evident, în cazul rezolvării unor probleme simple, elaborarea schemei logice poate să nu fie necesară, trecîndu-se direct la scrierea programului.

O schemă logică conține în general patru tipuri de operatori: operatori de intrare, operatori de ieșire, operatori aritmetici și operatori logici. Pentru elaborarea unei scheme logice se folosesc simboluri, indicate în tabelul 8. Simbolul, împreună cu conținutul înscris în interiorul său, formează un bloc. Conținutul blocului trebuie să fie cît mai sugestiv. O schemă logică se întocmește și se citește, de obicei, de la stînga la dreapta și de sus în jos.

Săgeata face legătura dintre blocuri în ordinea cerută de algoritm. Conectorul de pagină se folosește în interiorul aceleiași pagini pentru a evita intersecțiile de săgeți. Există un cerculeț de întrerupere și un cerculeț de reluare a săgeții. Pentru corespondența conectării, în interiorul cerculețului se înscrie un indicator, de exemplu o literă:

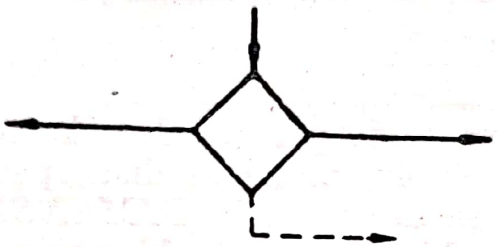
Acestea sînt valabile și pentru conectorul de trecere la o altă pagină. Delimitatorul de program conține, de obicei, cuvintele *START* sau *STOP*:

Scanned with OKEN Scanner

Nr. crt.	Denumirea simbolului	Forma grafică a simbolului	Utilizarea simbolului
1	Săgeata		Se utilizează la conectarea blocurilor unei scheme logice
2	Conector de pagină		Se utilizează la conectarea diferitelor puncte ale unei pagini de program
3	Conector de trecere la o altă pagină		Se utilizează la conectarea unor puncte ale schemei, care se află pe pagini diferite
4	Delimitator de program		Marchează începutul şi/sau sfîrşitul schemei logice
5	Bloc de intrare/ieşire		Se utilizează la introducerea informaţiei şi la extragerea rezultatelor
6	Bloc de decizie logică		Se utilizează la trecerea condiţionată în diverse puncte ale unei scheme
7	Blocul de calcul		În interiorul dreptunghiului se înscriu calculele care urmează să fie executate

Scanned with OKEN Scanner

Blocul de intrare/ieşire pune în evidenţă informaţia de intrare şi cea care este cerută la ieşire. Intrarea sau ieşirea se indică de obicei prin *CITEŞTE*, respectiv *SCRIE* sau *TIPĂREŞTE* sau se folosesc simbolurile specializate care se văd în tabelul 8: primul simbol pentru intrare, iar al doilea pentru ieşire. Blocul de decizie logică indică succesiunea calculelor după cum o condiţie logică este îndeplinită sau nu. El trebuie să aibă o intrare şi cel puţin două (cel mult trei) ieşiri.



Blocul de calcul conţine calculele în ordinea cerută de algoritm. Nu este necesar ca toate calculele cerute într-o anumită etapă a algoritmului să fie înscrise într-un singur bloc de calcul. Se pot utiliza mai multe blocuri de calcul, legate între ele prin săgeţi.

Elaborarea unei scheme logice este absolut necesară, atunci cînd problema este vastă şi cînd la analizarea şi rezolvarea ei lucrează eventual mai multe persoane. Prin legarea prin conectori a tuturor părţilor problemei, schema logică realizează un tot unitar.

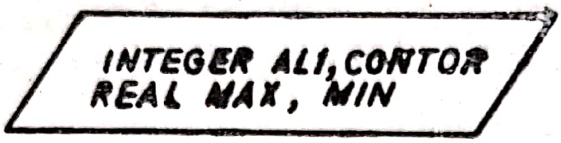
2.2.2. Scheme logice orientate spre un limbaj algoritmic de programare

De multe ori, cînd se are în vedere rezolvarea unei probleme, se precizează de la început limbajul algoritmic de programare în care urmează să fie scris programul pentru calculator. În acest sens, chiar schema logică este orientată spre limbajul ales. În cele ce urmează ne vom referi numai la limbajul FORTRAN prezentat în capitolul III. Simbolurile folosite sînt cele indicate în tabelul 8, dar conţinutul lor este specific limbajului algoritmic. Vom prezenta anticipat cîteva caracteristici ale unui program scris în FORTRAN pentru a vedea cum se reflectă acestea în schema logică orientată.

Pentru desemnarea variabilelor unei probleme se folosesc identificatori (şiruri de litere şi cifre, care încep cu o literă). În FORTRAN literele sînt litere mari $A - Z$. Prin construcţia limbajului, identificatorii care încep cu I, J, K, L, M, N sînt identificatori de variabile întregi, restul desemnînd variabile de tip real. De exemplu Il desemnează o variabilă întreagă, iar EXEMPL desemnează o variabilă reală.

Scanned with OKEN Scanner

Dacă dorim ca și EXEMPL să desemneze tot o variabilă întreagă, declarăm explicit acest lucru prin *INTEGER EXEMPL*. Deci, declararea explicită a variabilelor prin indicarea identificatorilor este bine să apară în schema logică la începutul blocului de intrare. De exemplu

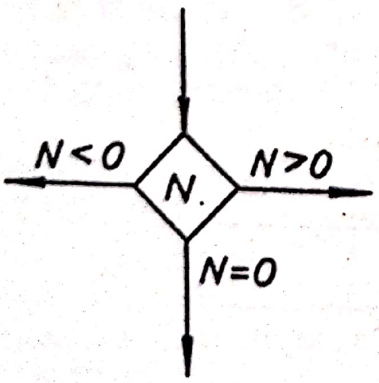


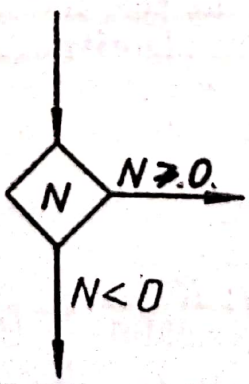
În FORTRAN se declară (după eventuala declarare a variabilelor) numele matricelor și dimensiunile lor. Numele se indică printr-un identificator de tablou, unde pentru identificator sînt valabile cele spuse mai sus. De exemplu, matricea distanțelor numită *DIST* cu 10 linii și 20 coloane se declară prin *DIMENSION DIST* (10, 20). Această declarație este bine să apară chiar în schema logică într-un bloc analog cu cel desenat anterior.

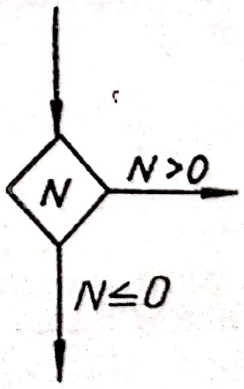
Pentru programe mai laborioase, se apelează subprograme întocmite de rezolvitor sau de altcineva și catalogate într-o bibliotecă a sistemului. Pentru apelarea și folosirea unor astfel de supragrame se folosește un simbol de forma

numit bloc de procedură. În interiorul lui se înscrie numele subprogramului.

Blocul de decizie logică în FORTRAN se utilizează avînd în vedere că ramificarea pe un anume drum se face după cum cantitatea înscrisă în bloc este strict negativă, zero, strict pozitivă sau combinații (permise) ale acestora:







Verificarea sau neverificarea unei condiții logice duce deci la un salt (condiționat) în program.

Există și posibilitatea unui salt necondiționat, de exemplu atunci cînd s-a terminat executarea unui anumit grup de calcule. Saltul necondiționat se poate menționa în schema logică prin includerea în blocul de calcule a instrucțiunii *GO TO K*, unde prin *K* se indică o marcă (etichetă) din program, adică un conector în schema logică. Pe lîngă acest fel de salt necondiționat, mai există un salt necondiționat realizat de instrucțiunea *GO TO* calculat [7].

Pentru a indica reluarea unor calcule sau a unui grup de calcule, există instrucțiunea de ciclare *DO*. Folosirea ei ușurează mult munca de programare. Scrierea unui program se poate face cu sau fără *DO*. Această alegere este bine să apară clar chiar din schema logică.

2.2.3. Exemple

Vom da ca exemple schemele logice de obținere a produsului scalar a doi vectori $a = (a_1, a_2, \ldots, a_n)$ și $b = (b_1, b_2, \ldots, b_n)$. Componentele vectorilor sînt numere reale, pe care calculatorul le citește de pe cartele. Înainte de a indica schemele logice, menționăm că notația $S = S + P$ are următorul sens: noua valoare a lui S (membrul din stînga) rezultă din valoarea veche a lui S (membrul din dreapta) la care se adaugă P. De exemplu, dacă valoarea inițială a lui S este $S = 0$, iar $P = 7$, atunci S devine pe rînd $S = 7$, $S = 14$, $S = 21$, etc.

Notăm cu S produsul scalar. Se știe că

$$S = a_1b_1 + a_2b_2 + \ldots + a_nb_n.$$

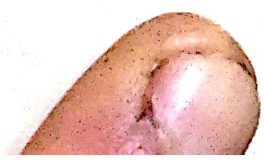
Notăm cu P produsul a două componente de același ordin a_ib_i. Presupunem că pe fiecare cartelă se află numai cîte o perche a_i, b_i. Calculatorul va citi perechile pe rînd, în măsura în care îi sînt necesare. (Se poate întocmi o schemă logică și presupunînd că se citesc de la început a și b în întregime).

Exemplul 1: schema logică generală.

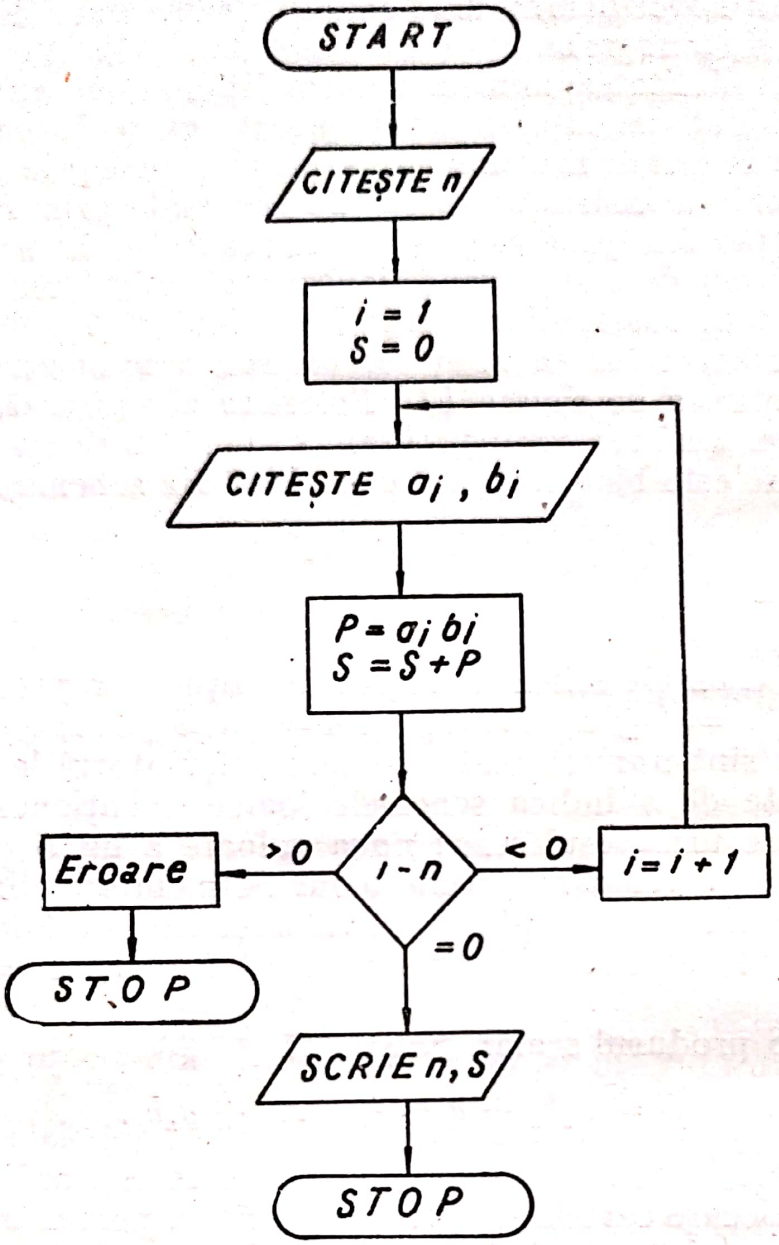
Pentru a face programul cît mai general, n se citește de pe o cartelă.

Exemplul 2: Schema logică orientată.

În acest exemplu presupunem că se știe că, pentru problema studiată, cea mai mare valoare a lui n este 250. De asemenea presupunem că de la început se introduc în memorie, de pe cartele, toate componen-



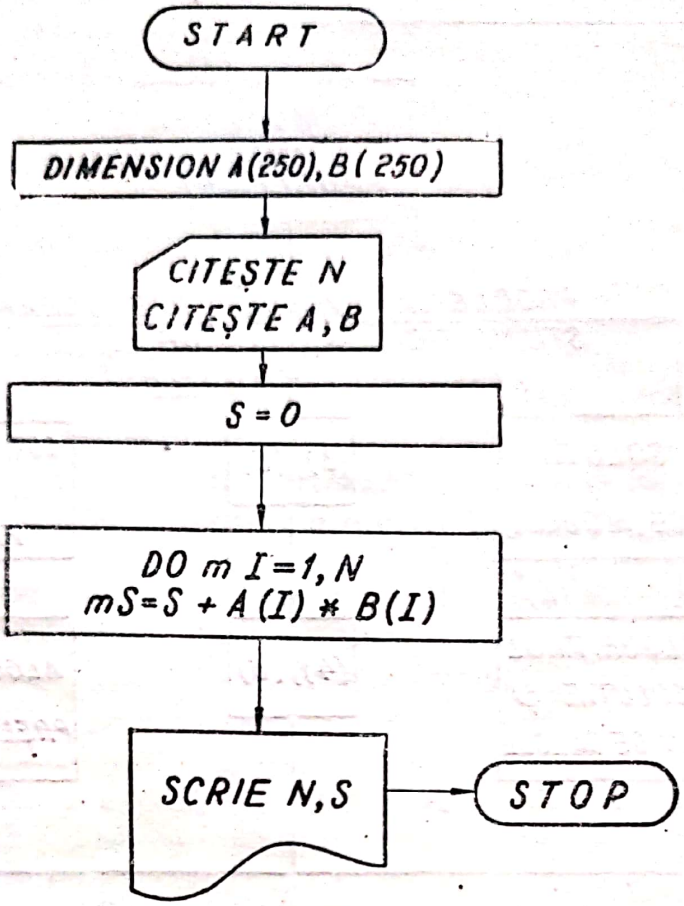
Scanned with OKEN Scanner



Ex. 1. Schema logică generală

tele lui *a*, apoi toate componentele lui *b*. Pentru că schema logică este FORTRAN-orientată, folosim *A* și *B* ca identificatori ai vectorilor *a* și *b*, iar pentru *n* folosim *N*. Pentru diverse probleme particulare, *N* poate fi mai mic de 250 și de aceea îl vom citi de pe o cartelă pentru fiecare problemă concretă. Indicarea lui 250 are ca scop alocarea spațiului în memorie pentru componentele lui *A* și *B*. Acest fapt îl vom indica în schema logică. De asemenea, vom indica instrucțiunea de ciclare *DO* pe care ne propunem să o folosim în scrierea programului.

Scanned with OKEN Scanner



Ex. 2. Schema logică orientată FORTRAN

În acest fel obținem următoarea schemă logică FORTRAN — orientată, în care prin *m* am notat o marcă, a cărei formă va fi precizată de programator la scrierea programului.

2.3. FAZELE REZOLVĂRII UNEI PROBLEME CU CALCULATORUL ELECTRONIC

Scopul prezentului paragraf este de a da o orientare în ce privește rezolvarea cu calculatorul electronic a diverselor probleme practice și de a oferi un ghid, pe baza căruia poate fi concepută rezolvarea. Vom schița unele aspecte legate de rezolvarea problemelor (aplicațiilor) cu calculatorul, de diverși utilizatori. Etapele de rezolvare (orientative) sînt prezentate în schema logică din figura 12. Numerotarea blocurilor corespunde cu cele ce urmează.

Scanned with OKEN Scanner

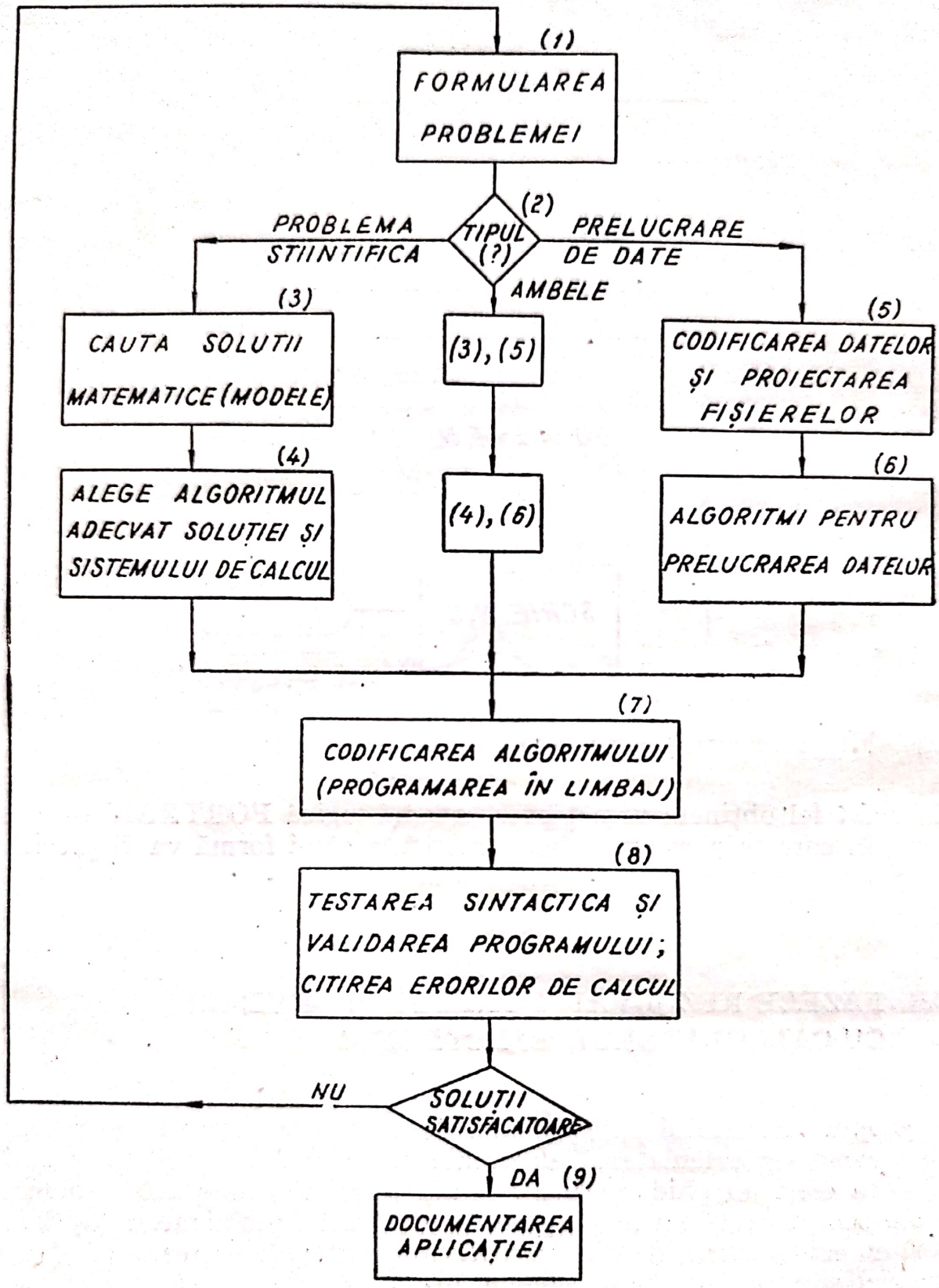


Fig. 12. Etapele rezolvării unei aplicații cu calculatorul

Scanned with OKEN Scanner

1. Pentru orice problemă ce urmează să fie rezolvată cu calculatorul, trebuie avut în vedere :

— întrebările la care trebuie să se răspundă ;

— natura informațiilor disponibile (numerice sau nenumerice = alfanumerice) ;

— rigoarea cu care trebuie formulate răspunsurile la anumite întrebări.

2. Printre aplicațiile (problemele) care se rezolvă cu calculatorul se disting două tipuri importante :

— aplicațiile (problemele) științifice ;

— prelucrările de date.

Problemele științifice se caracterizează prin aceea că utilizează puține date de intrare, în schimb necesită mult timp de calcul și multă memorie internă. Aplicațiile care constau în prelucrări de date se caracterizează prin existența unui număr mare de date de intrare și rezultate (la ieșire), care se prelucrează pe baza unor algoritmi relativ simpli, fără să necesite un timp prea mare de calcul. Prelucrările de date, de cele mai multe ori, necesită puțină memorie internă, dar multă memorie externă.

3. Dacă problema are un caracter științific, atunci se construiește sau se încadrează problema într-un model matematic și se alege algoritmul adecvat. Modelul trebuie să conțină : variabile de intrare (independente), parametri de intrare (constante furnizate de utilizator), variabile de ieșire (dependente), parametri de ieșire (constante) care rezultă din calcule, relații funcționale care leagă între ele variabilele și parametrii modelului și ipoteze de lucru. Valorile parametrilor de intrare trebuie cunoscute (eventual estimate pe baza informațiilor de care se dispune la formularea problemei). Rezolvarea unei probleme cuprinse într-un model constă în a exprima variabilele dependente funcție de cele independente, iar parametrii de ieșire funcție de parametrii de intrare ; parametrii de ieșire pot depinde și de variabilele de intrare/ieșire.

4. În cazul cînd se dispune de mai multe modele, sau pentru un model există mai mulți algoritmi, este necesar să se aleagă algoritmul care satisface condițiile :

— necesită un timp de calcul cît mai scăzut ;

— conduce la un program care nu depășește capacitatea memoriei calculatorului ;

— conduce la rezultate care sînt cît mai puțin afectate de erori.

5. Dacă problema de rezolvat are ca scop prelucrarea de date, trebuie să se construiască de asemenea un model care să conțină variabile și parametri cantitativi sau calitativi, după cum informațiile de prelucrat sînt numerice sau nenumerice. Informațiile trebuie să fie codificate. Codificarea se face printr-un cod acceptat de calculator,

Scanned with OKEN Scanner

în așa fel încît să fie cît mai ușor de manevrat la programare și la interpretarea rezultatelor. Codificarea este bine să fie făcută pe formulare speciale care urmează să fie aduse la echipamentele de pregătirea datelor (mașini de perforat, verificat, sortat, etc.).

Pe baza analizei volumului prezumtiv al informațiilor se trece apoi la construirea machetelor fișierelor (vezi cap. IV) de date, care trebuie prelucrate ținîndu-se cont de următoarele:

— fișierele să nu conțină informații de prisos;
— fișierele să folosească în mod eficient memoriile externe;
— sistemul de fișiere să fie adecvat configurației de calcul existente.

6. În funcție de întrebările formulate, trebuie găsiți algoritmii cei mai eficienți de prelucrare. Performanțele algoritmilor de prelucrare (rapiditate, exactitate, consum redus de memorie) sînt în strînsă legătură cu forma și mărimea fișierelor. Din această cauză etapele 5 și 6 se pot reconsidera de mai multe ori în vederea alegerii celei mai avantajoase variante.

7. Etapele precedente alcătuiesc *analiza problemei*. Etapele care urmează sînt legate de *programarea problemei pentru calculator*. Pentru programare, este importantă întocmirea unei liste complete a identificatorilor de variabile, adică a codurilor adoptate.

8. După scrierea programului într-un limbaj algoritmic de programare, acesta trece, înainte de a fi rulat pe calculator, prin următoarele etape: perforare (de obicei pe cartele), verificare și corectare sintactică.

Înainte de a introduce programul (pachetul de cartele) pe calculator, este bine ca utilizatorul să urmărească dacă cu ocazia perforării s-a respectat ordinea instrucțiunilor de pe formular. Prin acesta se observă și unele eventuale greșeli de perforare. După acesta, urmează *testarea sintactică* a programului prin introducerea lui la calculator. Greșelile sintactice indicate de calculator se vor corecta prin corectarea cartelelor greșite. Uneori aceste greșeli sînt formale, ca de exemplu lipsa unei paranteze de închidere, iar alteori sînt greșeli de fond care necesită intervenții în schema logică a problemei.

După ce programul nu mai are greșeli sintactice, se trece la executarea lui efectivă, fază în care se folosesc datele de intrare (în faza de testare sintactică a programului, datele de intrare nu sînt folosite). *Validarea programului* este o operație importantă în rezolvarea unei probleme cu calculatorul. Prin această operație se decide dacă programul, adus în faza de execuție, rezolvă corect problema pentru care el a fost întocmit. Validarea se face comparînd rezultatele produse de program cu rezultate — obținute pe baza acelorași valori ale parametrilor de intrare—a căror valabilitate este cunoscută sau cu rezultate obținute prin calcule făcute cu mîna. Aceasta înseamnă că

Scanned with OKEN Scanner

validarea programului se mai face prin rezolvarea unor *exemple de control*. După validarea programului, acesta va fi folosit pentru rezolvarea problemei cu date reale.

9. Pentru ca programul împreună cu soluţia problemei să poată fi utilizate şi în alte ocazii şi pentru ca ele să poată fi folosite cu uşurinţă şi de alţi utilizatori, este necesar ca rezolvarea problemei să se încheie prin nişte documente care compun dosarul problemei. Aceste documente sînt: enunţul problemei, lista variabilelor modelului, semnificaţiile şi codificarea lor, machetele fişierelor (dacă este cazul), algoritmul de calcul, instrucţiuni de utilizare a programului (indicaţii cu privire la intrări şi ieşiri, restricţii de memorie, menţiuni asupra erorilor, etc.). Instrucţiunile de utilizare trebuie să faciliteze ele singure folosirea programului de către alte persoane.

2.4. MODELE LINIARE

2.4.1. **Exemple de probleme care conduc la modele liniare**

În cele ce urmează vom enunţa descriptiv cîteva probleme şi apoi vom întocmi modele ale problemelor respective.

2.4.1.1. *Problema dietei.* Acestă problemă este una din primele probleme liniare formulate (Stigler — 1945). Ea constă în întocmirea unei diete dintr-un număr dat de alimente, care să satisfacă anumite cerinţe biologice şi care să fie cît mai ieftină.

Vom nota cu $A_1, \ldots, A_j, \ldots, A_n$ cele n alimente de care dispunem pentru întocmirea dietei, iar cu $P_1, \ldots, P_i, \ldots, P_m$ m principii biologice (vitamine, calorii, etc.) pe care dieta trebuie să le conţină. În ce cantitate? Dieta trebuie să le conţină cel puţin în cantităţile $b_1, \ldots b_i, \ldots, b_m$ pentru a satisface cerinţele biologice. Pentru a calcula costul dietei, trebuie cunoscut preţul unitar al fiecărui aliment. Fie $c_1, \ldots, c_j, \ldots, c_n$ aceste preţuri unitare. Cît să se consume din alimentele date, pentru ca cerinţele biologice să fie satisfăcute? Aceasta depinde de cantitatea de principiu nutritiv P_i ce se găseşte în unitatea de aliment A_j, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$. Fie a_{ij} această cantitate.

Cantităţile b_i, c_j, a_{ij} sînt numere reale cunoscute şi formează datele de intrare ale problemei. Necunoscutele problemei sînt cantităţile $x_1, \ldots, x_j, \ldots, x_n$ care trebuie să fie consumate din alimentele $A_1, \ldots A_j, \ldots, A_n$, aşa încît să fie satisfăcut scopul rezolvării problemei: dieta să satisfacă cerinţele biologice şi să fie cît mai ieftină. Pînă aici

Scanned with OKEN Scanner

am dat o descriere completă și corectă a problemei. Pentru o mai mare claritate, vom întocmi tabelul 9:

TABELUL 9

Principii nutritive	Alimente	Raţia biologică
	$A_1 \ldots A_j \ldots A_n$	
P_1	$a_{11} \ldots a_{1j} \ldots a_{1n}$	b_1
P_i	$a_{i1} \ldots a_{ij} \ldots a_{in}$	b_i
P_m	$a_{m1} \ldots a_{mj} \ldots a_{mn}$	b_m
Preţ	$c_1 \ldots c_j \ldots c_n$	Problemă de minim
Necunoscute	$x_1 \ldots x_j \ldots x_n$	

Modelul problemei, adică problema „pusă în ecuaţie", se prezintă în felul următor:
Să se determine x_j, $1 \leqslant j \leqslant n$, astfel ca

$$(1) \qquad \sum_{j=1}^{n} a_{ij} x_j \geqslant b_i, \quad 1 \leqslant i \leqslant m,$$

$$(2) \qquad x_j \geqslant 0, \quad 1 \leqslant j \leqslant n,$$

$$(3) \qquad \min \left(\sum_{j=1}^{n} c_j x_j \right).$$

Relaţiile (1) se numesc restricţii ale problemei, (2) se numesc condiţii de nenegativitate impuse variabilelor, iar $c_1 x_1 + \ldots + c_n x_n$ se numeşte funcţia de optimizat sau funcţia obiectiv. Valorile x_j, $1 \leqslant j \leqslant n$, care satisfac (1) şi (2) formează o *soluţie posibilă* a problemei, iar soluţia posibilă care minimizează funcţia obiectiv se numeşte *soluţie optimă*. (1), (2) şi (3) reprezintă un model matematic liniar.

2.4.1.2. *Organizarea raţională a producţiei.* O întreprindere are n maşini M_j, $1 \leqslant j \leqslant n$, pe care le foloseşte pentru a produce m tipuri de piese T_i, $1 \leqslant i \leqslant m$. Fiecare maşină M_j poate funcţiona a_j ore. Fiecare tip de piesă solicită maşinile un anumit timp. Planul de producţie prevede să se producă anumite cantităţi din fiecare tip de piese. Se cere

Scanned with OKEN Scanner

să se otimizeze activitatea în întreprindere astfel încît planul de producție să fie îndeplinit cu o cheltuială minimă sau cu un cîștig maxim.

Vom trata problema sub două aspecte:

1. Presupunem că la producerea unei piese participă o singură mașină.

Pentru a putea scrie modelul matematic al problemei, vom nota prin:

t_{ij} numărul de unități de timp cît funcționează mașina M_j pentru a produce o piesă de tipul T_i, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$;

b_i numărul de piese de tipul T_i cerut de planul de producție;

c_{ij} cheltuieli de producție: suma cheltuită pentru producerea unei piese T_i pe mașina M_j;

x_{ij} necunoscutele problemei: numărul de piese T_i care vor fi produse de mașina M_j.

Intocmim tabelul 10:

TABELUL 10

Tipuri de piese	Mașini					Planul de producție
	M_1	.....	M_j	.....	M_n	
T_1	t_{11} c_{11}	. . . .	t_{1j} c_{1j}	. . . . .	t_{1n} c_{1n}	b_1
. . . .	. . . .		. . . .		. . . .	. . . .
T_i	t_{i1} c_{i1}	. . . .	t_{ij} c_{ij}	. . . . .	t_{in} c_{in}	b_i
. . . .	. . . .		. . . .		. . . .	. . . .
T_m	t_{m1} c_{m1}	. . . .	t_{mj} c_{mj}	. . . . .	t_{mn} c_{mn}	b_m
Timp de funcționare	a_1	. . . .	a_j	. . . . .	a_n	Problema de minim

Scanned with OKEN Scanner

Ecuaţiile, care constituie modelul matematic al problemei, se scriu în felul următor :

Cantităţile de piese produse trebuie să fie cel puţin în cantitatea cerută de plan, adică

$$\sum_{j=1}^{n} x_{ij} \geqslant b_i, \qquad i = 1, \ldots, m. \tag{4}$$

Să nu fie depăşit timpul de funcţionare al fiecărei maşini, adică

$$\sum_{i=1}^{m} t_{ij} x_{ij} \leqslant a_j, \qquad j = 1, \ldots, n. \tag{5}$$

Numărul de piese produse să fie nenegativ

$$x_{ij} \geqslant 0, \; i = 1, \ldots, m; \; j = 1, \ldots, n. \tag{6}$$

Costul de producţie să fie cît mai mic, adică

$$\min \left(\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \right). \tag{7}$$

2. Presupunem că, la producerea unei piese T_i, participă mai multe maşini (eventual toate). Dacă notăm cu c_i, $1 \leqslant i \leqslant m$, beneficiul realizat de întreprindere prin vînzarea unei piese de tipul T_i şi dacă notăm necunoscutele problemei cu x_i, $1 \leqslant i \leqslant m$, cu semnificaţia că x_i reprezintă numărul de piese de tip T_i produs de întreprindere, atunci ecuaţiile problemei sînt :

$$x_i \geqslant b_i, \qquad i = 1, \ldots, m, \tag{4'}$$

$$\sum_{i=1}^{m} t_{ij} x_i \leqslant a_j, \qquad j = 1, \ldots, n, \tag{5'}$$

$$x_i \geqslant 0, \qquad i = 1, \ldots, m, \tag{6'}$$

$$\max \left(\sum_{i=1}^{m} c_i x_i \right). \tag{7'}$$

În ambele cazuri, modelele conţin $m + n$ restricţii. În cazul al doilea, dacă se face schimbarea de variabile $x_i = b_i + y_i$ cu $y_i \geqslant 0$, $1 \leqslant i \leqslant m$, numărul variabilelor se reduce la n. Într-adevăr, obţinem

$$\sum_{i=1}^{m} t_{ij}(b_i + y_i) = \sum_{j=1}^{m} t_{ij} y_i + \sum_{i=1}^{m} t_{ij} b_i \leqslant a_j, \; 1 \leqslant j \leqslant n.$$

Scanned with OKEN Scanner

Notăm $a'_j = a_j - \sum_{i=1}^{m} t_{ij} b_i$, $1 \leqslant j \leqslant n$. Atunci, $(5'_j)$ devine

$$(5'') \quad \sum_{i=1}^{m} t_{ij}\, y_i \leqslant a'_j, \qquad j = 1, \ldots, n,$$

$$(6'') \quad y_i \geqslant 0, \qquad i = 1, \ldots, n.$$

Funcţia de optimizat devine

$$\sum_{i=1}^{m} c_i\, x_i = \sum_{i=1}^{m} c_i\, y_i + \sum_{i=1}^{m} c_i\, b_i.$$

Pentru că termenul al doilea din sumă este constant, relaţia (7') devine

$$(7'') \quad \max \left(\sum_{i=1}^{m} c_i\, y_i \right).$$

Modelul (5''), (6''), (7'') conţine n restricţii, m condiţii de nenegativitate şi o funcţie liniară de maximizat.

2.4.1.3. *Aproximarea Cebîşev a unui sistem de funcţii liniare.* Fie m funcţii liniare $f_i : R^m \to R$, de forma

$$f_i(x) = a_{i1}\, x_1 + \ldots + a_{in}\, x_n + b_i, \qquad 1 \leqslant i \leqslant m,$$

unde a_{ij}, b_i, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$, sînt numere reale date. Problema aproximării Cebîşev a acestui sistem de m funcţii liniare constă în găsirea unui punct $x^* \in R^n$, astfel încît

$$(8) \quad \max_{1 \leqslant i \leqslant m} |f_i(x^*)| = \inf_{x \in R^n} \max_{1 \leqslant i \leqslant m} |f_i(x)|.$$

În rezolvarea acestei probleme, apar dificultăţi foarte mari pentru că în (8) apare inferiorul calculat după toţi $x \in R^n$, ceea ce practic este imposibil. Din această cauză, se pune problema dată sub o formă echivalentă, pentru care scrierea modelului este foarte simplă. Pentru aceasta, fie g o funcţie de o variabilă şi anume $g(x_{n+1}) = x_{n+1}$. Vom minimiza funcţia g în felul următor:

$$(9) \quad \min\ (x_{n+1}),$$

cu restricţiile

$$(10) \quad |f_i(x)| \leqslant x_{n+1}, \quad i = 1, \ldots, m.$$

Se demonstrează că problema (8) este echivalentă cu problema liniară (9), (10). Rezolvarea problemei (9), (10) se face prin metodele obişnuite ale programării liniare.

Scanned with OKEN Scanner

2.4.1.4. *Problema transporturilor.* Ea a fost enunțată pentru prima dată de Hitchcock în 1941. Este o problemă cu o structură particulară, relativ simplă, pentru care s-au imaginat metode specifice de rezolvare.

Problema se formulează în felul următor: de la m centre de producție P_i, $1 \leqslant i \leqslant m$, un anumit produs divizibil trebuie transportat la n centre de consum Q_j, $1 \leqslant j \leqslant n$. În P_i este disponibilă cantitatea a_i, iar în Q_j se cere cantitatea b_j. Costul transportului de la P_i la Q_j este c_{ij}, cunoscut. Problema cere să se întocmească un plan de transport, care să satisfacă cererea, să nu depășească disponibilul și să fie cît mai ieftin posibil. Notăm cu x_{ij}, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$ necunoscutele problemei, adică cantitatea care trebuie transportată din P_i în Q_j. Întocmim tabelul 11.

TABELUL 11

Centre de producție	Centre de consum					Cantitățile disponibile
	Q_1	. . .	Q_j	. . .	Q_n	
P_1	c_{11}	. . .	c_{1j}	. . .	c_{1n}	a_1
.	.					.
.	.					.
.	.					.
P_i	c_{i1}	. . .	c_{ij}	. . .	c_{in}	a_i
.	.					.
.	.					.
.	.					.
P_m	c_{m1}	. . .	c_{mj}	. . .	c_{mn}	a_m
Cantitățile cerute	b_1	. . .	b_j	. . .	b_n	Problema de minim

Suma cantităților transportate nu trebuie să depășească disponibilul din fiecare centru de producție, adică

$$(11) \quad \sum_{j=1}^{n} x_{ij} \leqslant a_i, \qquad i = 1, \ldots, m.$$

Suma cantităților transportate trebuie să asigure cel puțin cererea din fiecare centru de consum, adică

$$(12) \quad \sum_{i=1}^{m} x_{ij} \geqslant b_j, \qquad j = 1, \ldots, n.$$

Cantitățile transportate să fie nenegative, adică

$$(13) \quad x_{ij} \geqslant 0, \qquad i = 1, \ldots, m; \qquad j = 1, \ldots, n.$$

Costul transportului să fie minim, adică să se realizeze

$$(14) \quad \min \left(\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \right).$$

Scanned with OKEN Scanner

Pentru ca problema să aibă cel puţin o soluţie posibilă, trebuie ca

$$(15)\quad \sum_{i=1}^{m} a_i \geqslant \sum_{j=1}^{n} b_j.$$

Dacă cererea depăşeşte oferta, atunci evident rezolvarea problemei este imposibilă. Fără a restrînge generalitatea problemei, prin introducerea unui consumator fictiv Q_{n+1}, în (15) se poate considera egalitate totdeauna.

2.4.2. Algoritme pentru modele liniare

2.4.2.1. *Algoritmul simplex.* În acest paragraf, totdeauna un vector $x \in R^n$ este considerat un vector coloană. Pentru că este mai comodă scrierea pe linie, atunci cînd vom indica componentele lui x, vom folosi vectorul transpus

$$x = (x_1, \ldots, x_n)',$$

unde prin " ' " am notat operaţia de transpunere.

Fie următoarea problemă liniară scrisă matriceal:

$$(16)\quad Ax = b,$$

$$(17)\quad x \geqslant 0,$$

$$(18)\quad \max c'x,$$

unde $x \in R^n$, $x = (x_1, \ldots, x_n)'$ este vectorul necunoscutelor problemei, $b \in R^m$, $b = (b_1, \ldots, b_m)'$ este un vector ale cărui elemente sînt numere reale date. Matricea A, de elemente a_{ij}, are m linii şi n coloane, iar $c \in R^n$ are componentele $c_j \in R$ date.

Operaţia de substituire a unui vector cu alt vector. Se spune că sistemul de vectori $a_i \in R^m$, $1 \leqslant i \leqslant s$, este *liniar independent*, dacă din combinaţia liniară

$$\sum_{i=1}^{s} \lambda_i a_i = 0, \qquad (0 \in R^m),$$

rezultă toţi λ_i nuli ($1 \leqslant i \leqslant s$). Dacă rezultă cel puţin un λ_i nenul, sistemul se numeşte *liniar dependent.*

Fie vectorii $a_i \in R^m$, $1 \leqslant i \leqslant s$, liniar independenţi şi fie vectorii $b_j \in R^m$, $1 \leqslant j \leqslant n$, liniar dependenţi de a_i, adică b_j se exprimă ca o combinaţie liniară a vectorilor a_i, cu coeficienţii ε_{ij}, adică

$$b_j = \sum_{i=1}^{s} \varepsilon_{ij} a_i, \qquad j = 1, \ldots, n.$$

Scanned with OKEN Scanner

Am notat prin $a_i = (a_{1i}, a_{2i}, \ldots, a_{mi})'$, $1 \leqslant i \leqslant s$ și $b_j = (b_{1j}, b_{2j}, \ldots, b_{mj})'$, $1 \leqslant j \leqslant n$.

Vrem să înlocuim pe unul din vectorii a_i, de exemplu pe a_l, $1 \leqslant l \leqslant s$, cu vectorul b_k, $1 \leqslant k \leqslant n$, astfel încît sistemul de vectori

$$a_1, a_2, \ldots, a_{l-1}, b_k, a_{l+1}, \ldots, a_s \tag{19}$$

să rămînă liniar independent și să obținem o exprimare a vectorilor b_j, $1 \leqslant j \leqslant n$, cu ajutorul noului sistem (19). Ne întrebăm: a) care este condiția ca a_l să poată fi înlocuit cu b_k astfel încît (19) să rămînă liniar independent și b) dacă schimbarea se poate face, cum se modifică coeficienții exprimării lui b_j cu vectorii din (19). Dacă notăm coeficienții noi ai exprimării cu ε'_{ij}, întocmim tabelul 12.

TABELUL 12

	b_1	b_2	$\ldots$	b_j	$\ldots$	b_k	$\ldots$	b_n
a_1	ε_{11}	ε_{12}	$\ldots$	ε_{1j}	$\ldots$	ε_{1k}	$\ldots$	ε_{1n}
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
a_i	ε_{i1}	ε_{i2}	$\ldots$	ε_{ij}	$\ldots$	ε_{ik}	$\ldots$	ε_{in}
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
a_{l-1}								
a_l	ε_{l1}	ε_{l2}	$\ldots$	ε_{lj}	$\ldots$	ε_{lk}	$\ldots$	ε_{ln}
a_{l+1}	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
a_s	ε_{s1}	ε_{s2}	$\ldots$	ε_{sj}	$\ldots$	ε_{sk}	$\ldots$	ε_{sn}
a_1	ε'_{11}	ε'_{12}	$\ldots$	ε'_{1j}	$\ldots$	ε'_{1k}	$\ldots$	ε'_{1n}
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
a_i	ε'_{i1}	ε'_{i2}	$\ldots$	ε'_{ij}	$\ldots$	ε'_{ik}	$\ldots$	ε'_{in}
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
a_l								
b_k	ε'_{l1}	ε'_{l2}	$\ldots$	ε'_{lj}	$\ldots$	ε'_{lk}	$\ldots$	ε'_{ln}
a_{l+1}	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$
a_s	ε'_{s1}	ε'_{s2}	$\ldots$	ε'_{sj}	$\ldots$	ε'_{sk}	$\ldots$	ε'_{sn}

Scanned with OKEN Scanner

Răspunsul la cele două întrebări de mai sus este dat de

Teorema 1. a) Condiția necesară și suficientă ca sistemul (19) să fie liniar independent este ca $\varepsilon_{lk} \neq 0$ și b) în acest caz, toți vectorii b_j se exprimă ca o combinație liniară de acești vectori, cu coeficienții

$$\varepsilon'_{ij} = \varepsilon_{ij} - \frac{\varepsilon_{ik}\varepsilon_{lj}}{\varepsilon_{lk}}, \quad i \neq l, \quad i = 1,\ldots,s, \quad j = 1,\ldots,n; \tag{20}$$

$$\varepsilon'_{lj} = \frac{\varepsilon_{lj}}{\varepsilon_{lk}}, \qquad j = 1,\ldots,n. \tag{21}$$

Teoremele de bază ale algoritmului simplex. Vom nota problema (16), (17), (18) prin $\mathcal{A}$. Se numește rang al matricei A ordinul celui mai mare determinant nenul, extras din matricea A. Se notează $r = \text{rang } A$. Vom nota cu a_j, $1 \leqslant j \leqslant n$, coloanele matricei A. Soluția posibilă x a lui $\mathcal{A}$ se numește soluția posibilă de bază, dacă mulțimea vectorilor a_j, asociați lui $x_j \neq 0$, este liniar independentă. Dacă această mulțime are un număr de vectori egali cu rang A, atunci soluția posibilă de bază se numește nedegenerată, iar mulțimea însăși se numește bază nedegenerată.

Se pune problema găsirii unei soluții posibile de bază inițiale. Pentru aceasta, fiecărei restricții a lui $\mathcal{A}$ i se asociază o variabilă auxiliară. De exemplu $a_{i1}x_1 + \ldots + a_{in}x_n \leqslant b_i$ sau $a_{i1}x_1 + \ldots + a_{in}x_n = b_i$, $1 \leqslant i \leqslant m$, devin $a_{i1}x_1 + \ldots a_{in}x_n + y_i = b_i$, $1 \leqslant i \leqslant m$. Atunci, o soluție posibilă de bază a problemei $\mathcal{A}$ transformată în acest fel, este dată de $x_1 = 0, \ldots, x_n = 0, y_1 = b_1, \ldots, y_m = b_m$, adică $x = 0$, $y = b$.

Notăm cu $u_i = (0,\ldots,0,1,0,\ldots,0)' \in R^m$ vectorul unitar care are numai componenta de ordinul i egală cu 1, $1 \leqslant i \leqslant m$. Rezolvarea lui $\mathcal{A}$ începe în următorul mod indicat în tabelul 13:

TABELUL 13

Baza	a_1 .... a_j .... a_n	b	u_1 .... u_i .... u_m
u_1	a_{11} . . . . . a_{1j} . . . a_{1n}	b_1	1. . . . 0. 0
.	. . . .	.	. . . .
.	. . . .	.	. . . .
.	. . . .	.	. . . .
u_i	a_{i1} a_{ij} a_{in}	b_i	0 1 0
.	. . . .	.	. . . .
.	. . . .	.	. . . .
.	. . . .	.	. . . .
u_m	a_{m1} . . . . . a_{mj} . . . a_{mn}	b_m	0 . . . . 0 1

Scanned with OKEN Scanner

În tabelul 13, cantităţile a_{ij} sînt elementele matricei iniţiale A. După întocmirea acestui tabel, urmează înlocuirea a cîte un u_i cu cîte un a_j, atît timp cît se poate, după cum arată teorema 1. Operaţia de substituire se poate repeta de atîtea ori cît este rang A. Presupunem că rang $A = m$. După m paşi, obţinem tabelul 14, completat cu o linie şi o coloană rezervată vectorului $c \in R^n$:

TABELUL 14

	Baza	$c_1 \ \dots \ c_m$	$c_{m+1} \ \dots \ c_n$	b	$p_1 \ \dots \ p_i \ \dots \ p_m$
		$a_1 \ \dots \ a_m$	$a_{m+1} \ \dots \ a_n$		$u_1 \ \dots \ u_i \ \dots \ u_m$
c_1	a_1	$1 \ \dots \ 0$	$\varepsilon_{1,m+1} \ \dots \ \varepsilon_{1n}$	x_1	$\alpha_{11} \ \dots \ \alpha_{1i} \ \dots \ \alpha_{1m}$
$\cdot$	$\cdot$	$\cdot \ \ \cdot$	$\cdot \ \ \cdot$	$\cdot$	$\cdot \ \ \cdot \ \ \cdot$
$\cdot$	$\cdot$	$\cdot \ \ \cdot$	$\cdot \ \ \cdot$	$\cdot$	$\cdot \ \ \cdot \ \ \cdot$
$\cdot$	$\cdot$	$\cdot \ \ \cdot$	$\cdot \ \ \cdot$	$\cdot$	$\cdot \ \ \cdot \ \ \cdot$
c_i	a_i	$0 \ \dots \ 0$	$\varepsilon_{i,m+1} \ \dots \ \varepsilon_{in}$	x_i	$\alpha_{i1} \ \dots \ \alpha_{ii} \ \dots \ \alpha_{im}$
$\cdot$	$\cdot$	$\cdot \ \ \cdot$	$\cdot \ \ \cdot$	$\cdot$	$\cdot \ \ \cdot \ \ \cdot$
$\cdot$	$\cdot$	$\cdot \ \ \cdot$	$\cdot \ \ \cdot$	$\cdot$	$\cdot \ \ \cdot \ \ \cdot$
$\cdot$	$\cdot$	$\cdot \ \ \cdot$	$\cdot \ \ \cdot$	$\cdot$	$\cdot \ \ \cdot \ \ \cdot$
c_m	a_m	$0 \ \dots \ 1$	$\varepsilon_{m,m+1} \ \dots \ \varepsilon_{mn}$	x_m	$\alpha_{m1} \ \dots \ \alpha_{mi} \ \dots \ \alpha_{mm}$

Valorile p_i ale vectorului $p \in R^m$ vor fi precizate în cele ce urmează. Se demonstrează că $x = (x_1, \dots, x_m, 0, \dots 0)' \in R^n$ formează o soluţie posibilă de bază a lui $\mathcal{A}$. Dar apar întrebările: cum recunoaştem dacă x găsit în modul arătat mai sus este soluţia optimă (dacă $\mathcal{A}$ are soluţie optimă), sau cum recunoaştem că $\mathcal{A}$ nu are soluţie optimă? La aceste întrebări răspund următoarele trei teoreme. Pentru enunţarea lor, notăm cu $\mathcal{B}$ mulţimea indicilor vectorilor din bază (în tabelul 14 am considerat că primii m vectori a_i înlocuiesc toţi vectorii u_i. Acest lucru nu este obligatoriu, ci a_k poate înlocui un vector oarecare u_e şi de asemenea, se poate ca numai o parte din u_i să fie înlocuiţi cu a_j, aceasta cînd rang $A < m$. Se consideră totdeauna $m < n$). Mai notăm cu

$$z_j = \sum_{i \in \mathcal{B}} c_i \varepsilon_{ij}, \qquad j = 1, \dots, n. \tag{22}$$

În cazul tabelului 14, $\mathcal{B} = (1, \dots, m)$.

Teorema 2. Dacă toţi $z_j - c_j \geqslant 0$, $1 \leqslant j \leqslant n$, atunci soluţia posibilă de bază x, găsită în tabelul 13, este o soluţie optimă a lui $\mathcal{A}$.

Teorema 3. Dacă există k, $1 \leqslant k \leqslant m$, astfel ca $z_k - c_k < 0$ şi $\varepsilon_{ik} \leqslant 0$ pentru toţi $1 \leqslant i \leqslant m$, atunci $\mathcal{A}$ nu are soluţie optimă.

Scanned with OKEN Scanner

Teorema 4. Dacă există k, $1 \leqslant k \leqslant m$, astfel ca $z_k - c_k < 0$, dar există cel puţin un $\varepsilon_{ik} > 0$, $1 \leqslant i \leqslant m$, atunci se poate obţine o soluţie posibilă de bază, pentru care valoarea funcţiei de optimizat este mai mare, adică soluţia posibilă de bază se poate îmbunătăţi. Îmbunătăţirea ei se realizează introducînd în bază pe a_k şi scoţînd din bază pe a_l (a_l poate fi un u_l), unde indicele l se determină din condiţia:

$$\min_{i/\varepsilon_{ik}>0} \frac{x_i}{\varepsilon_{ik}} = \frac{x_l}{\varepsilon_{lk}}. \tag{23}$$

Referitor la componentele p_i ale lui p, facem următoarea observaţie: $p_i = 0$ dacă restricţia i iniţială (înainte de adăugarea variabilei auxiliare y_i) este inegalitate şi $p_i = -1$ sau $p_i = +1$, cînd restricţia i iniţială este egalitate şi problema $\mathcal{A}$ este de maxim, respectiv de minim.

O problemă de minim se transformă într-o problemă de maxim schimbînd c_j în $-c_j$ şi ţinînd cont că

$$\max f(x) = -\min [-f(x)],$$

unde prin $f(x)$ am notat funcţia de optimizat.

Algoritmul simplex pentru problema de maxim. Algoritmul cuprinde următorii paşi:

1. Se calculează diferenţele $z_j - c_j$ pentru toţi $1 \leqslant j \leqslant n$. Dacă toate diferenţele sînt nenegative, atunci soluţia posibilă de bază obţinută este o soluţie optimă; dacă există diferenţe negative, atunci:

2. Se alege diferenţa negativă cea mai mare în valoare absolută, de exemplu $z_k - c_k < 0$. Dacă $\varepsilon_{ik} \leqslant 0$ pentru toţi $1 \leqslant i \leqslant m$, atunci problema $\mathcal{A}$ nu are soluţie optimă. Dacă există $\varepsilon_{ik} > 0$, atunci:

3. Vectorul a_k intră în bază în locul lui a_l, unde l se determină calculînd

$$\min_{i/\varepsilon_{ik}>0} \frac{x_i}{\varepsilon_{ik}} = \frac{x_l}{\varepsilon_{lk}}.$$

4. Se calculează noile componente ale vectorilor a_j (coeficienţii ε'_{ij}) şi noile valori x'_i şi se trece la pasul 1, cercetînd noile valori $z'_j - c_j$ (c_j rămîn neschimbate).

Valorile x'_i şi $z'_j - c'_j$ se calculează după formule analoge cu formulele (20) şi (21), adică după „regula dreptunghiului". Demonstraţiile teoremelor enunţate se găsesc, de exemplu, în [4], [6], [11].

2.4.2.2. *Găsirea unei soluţii posibile pentru o problemă de transport.* Fie $\mathcal{T}$ o problemă de transport, definită de (11)—(15). Ca şi la $\mathcal{A}$,

Scanned with OKEN Scanner

găsirea unei soluții posibile inițiale de bază este punctul de plecare pentru găsirea soluției optime. Vom indica două asemenea metode.

Metoda unghiului de Nord-Vest. Notăm cu $X = (x_{ij})$ un plan de transport, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$. Prin metoda N—V, componentele lui X se determină în felul următor:

1. Se găsește $\min(a_1, b_1)$;

a. dacă $\min(a_1, b_1) = a_1$, se pune $x_{11} = a_1$ și $x_{1j} = 0$, $j = 2, \ldots, n$; se calculează noile cantități disponibile și noile cantități cerute, adică $a_1' = 0$, $a_i' = a_i$, $i = 2, \ldots, m$, $b_1' = b_1 - x_{11}$, $b_j' = b_j$, $j = 2, \ldots, n$ și se trece la pasul 2;

b. dacă $\min(a_1, b_1) = b_1$, se pune $x_{11} = b_1$ și $x_{i1} = 0$, $i = 2, \ldots, m$; se calculează noile cantități disponibile și noile cantități cerute, adică $a_1' = a_1 - x_{11}$, $a_i' = a_i$ și $b_1' = 0$, $b_j' = b_j$, $j = 2, \ldots, n$; se trece la pasul 2;

c. dacă $\min(a_1, b_1) = a_1 = b_1$, se alege arbitrar unul din cazurile a sau b după care se trece la pasul 2;

2. Se găsește minimul dintre prima valoare a_i' nenulă și prima valoare b_j' nenulă și se procedează analog cu a, b sau c de la pasul 1.

În această metodă se vede că valorile c_{ij} nu influențează cu nimic soluția x.

Metoda elementului minimal. În această metodă soluția x depinde de c_{ij}.

1. Se găsește cel mai mic element din matricea $C = (c_{ij})$, pentru toți i și j, adică $\min\limits_{i,j} c_{ij}$. Dacă minimul nu este unic, se ia arbitrar unul din ei.

Fie $\min\limits_{i,j} c_{ij} = c_{lk}$. Se găsește $\min(a_l, b_k)$;

a. dacă $\min(a_l, b_k) = a_l$, se pune $x_{lk} = a_l$ și $x_{lj} = 0$, $j = 1, \ldots, n$, $j \neq k$; se calculează noile cantități disponibile și noile cantități cerute, adică $a_i' = a_i$, $i \neq 1$, $a_l' = 0$ și $b_j' = b_j$, $j \neq k$, $b_k' = b_k - x_{lk}$; se trece la pasul 2;

b. dacă $\min(a_l, b_k) = b_k$, se pune $x_{lk} = b_k$ și $x_{ik} = 0$, $i = 1, \ldots m$, $i \neq 1$; se calculează noile cantități disponibile și noile cantități cerute, adică $a_i' = a_i$, $i \neq 1$, $a_l' = a_l - x_{lk}$ și $b_j' = b_j$, $j \neq k$, $b_k' = 0$; se trece la pasul 2;

c. dacă $\min(a_l, b_k) = a_l = b_k$, se alege arbitrar unul din cazurile a sau b și se trece la pasul 2;

Scanned with OKEN Scanner

2. În matricea inițială C se anulează linia l sau coloana k, după cum s-a trecut la pasul 2 din cazul a sau din cazul b, obținîndu-se matricea C' (apostroful nu înseamnează transpunere); se găsește cel mai mic element din matricea C' și se procedează analog cu a, b și c de la pasul 1.

2.5. MODELE NELINIARE

2.5.1. Exemple de probleme care conduc la modele neliniare

În paragraful precedent, modelele prezentate au funcția obiectiv liniară, pentru că am presupus că prețul de cumpărare (la problema dietei), sau prețul de vînzare (la varianta a doua a organizării raționale a producției) depinde liniar de cantitatea cumpărată sau de cantitatea vîndută. În realitate, prețurile nu sînt constante așa cum s-a presupus. O presupunere mai realistă este aceea că prețul de cumpărare c_j scade o dată cu creșterea cantității x_j cumpărate, de exemplu, în următorul mod $c_j = p_j - q_j x_j$, $p_j \geqslant 0$, $q_j \geqslant 0$. Atunci, funcția de optimizat a problemei dietei devine:

$$\max \left(\sum_{j=1}^{n} p_j x_j - \sum_{j=1}^{n} q_j x_j^2 \right). \tag{22}$$

Această funcție este pătratică și deci modelul problemei dietei devine un model pătratic (neliniar). Dacă și restricțiile problemei devin neliniare, atunci modelul se numește neliniar. Dacă prețul de cumpărare c_j depinde neliniar de x_j, de exemplu $c_j = p_j + q_j x_j + r_i x_j^2$, atunci funcția de optimizat devine de gradul al treilea și deci modelul problemei este neliniar, etc.

Același fel de presupuneri pot fi făcute la toate modelele liniare din paragraful precedent. Într-adevăr, la problema transporturilor de exemplu, se poate presupune că prețul c_{ij} al transportului unității de produs de la P_i la Q_j nu este constant, ci depinde într-un anume mod de cantitatea transportată, adică c_{ij} este o funcție $g(x_{ij})$. În acest caz, modelul devine pătratic sau chiar neliniar.

Pentru astfel de probleme, algoritmul simplex nu se mai poate aplica sub forma prezentată în § 2.4.2.1. În § 2.5.2 vom arăta cîteva aspecte ale rezolvării problemelor pătratice și neliniare.

Scanned with OKEN Scanner

2.5.2. Asupra rezolvării problemelor pătratice și a problemelor neliniare

Se face o distincție netă între problemele liniare, pătratice și neliniare. Se numesc *probleme liniare* numai acele probleme pentru care restricțiile sînt liniare, adică au forma $a_i x \leqslant b_i$ (sau $a_i x = b_i$ sau $a_i x \geqslant b_i$) și funcția obiectiv $f(x)$ este liniară. Se numesc *probleme pătratice* numai acele probleme care au restricții liniare (sub formele de mai sus) și funcția obiectiv pătratică. Toate celelalte probleme de optim, care nu sînt liniare sau pătratice, sînt *problene neliniare*. De exemplu, problema care are $f(x)$ pătratică și cel puțin o restricție pătratică este o problemă neliniară, chiar dacă toate celelte restricții sînt liniare.

Rezolvarea fiecăruia din cele trei tipuri de probleme de mai sus se face prin algoritmi specializați sau prin algoritmi speciali adaptați. De exemplu, pentru problemele liniare există algritmul simplex. Pentru problemele pătratice și problemele neliniare sub forma lor cea mai generală nu există, pînă în prezent, algoritmi de rezolvare. Algoritmii care există se referă la o clasă specială de probleme și anume *clasa problemelor convexe*. Pentru ca o problemă să aparțină acestei clase, trebuie ca atît funcția obiectiv, cît și funcțiile restricții să fie convexe. O funcție $f(x)$ se numește convexă, dacă oricare ar fi x și y din domeniul său de definiție și oricare are fi $0 < \lambda < 1$,

$$f[\lambda x + (1 - \lambda)y] \leqslant \lambda f(x) + (1 - \lambda)f(y). \tag{23}$$

Funcțiile convexe au o proprietate remarcabilă și anume aceea că minimul local este un mimim global. (Analog, pentru funcțiile concave : maximul local este un maxim global. O funcție f este concavă dacă $-f$ este convexă). Cum funcțiile liniare sînt și convexe și concave, o problemă pătratică este convexă, dacă funcția obiectiv este convexă.

Pentru problemele pătratice convexe există mulți algoritmi de rezolvare. Dintre acești algoritmi menționăm algoritmul lui Beale [1], algoritmul lui Wolfe [13] sub forma lungă și sub forma scurtă, algoritmul lui Hildreth și D'Esopo, algoritmul lui Theil-Van de Panne. Algoritmul lui Wolfe este o adaptare a algoritmului simplex pentru problema pătratică convexă. Algoritmul lui Theil-Van de Panne este o metodă combinatorială, aplicabilă în cazul cînd funcția obiectiv este strict convexă. (O funcție este strict convexă dacă în (23) apare inegalitatea strictă $<$). O descriere completă a algoritmilor de mai sus se găsește în [4] sau [9].

Pentru problemele neliniare convexe menționăm algoritmul lui Kelley. Acest algoritm reduce rezolvarea problemei neliniare convexe la rezolvarea unei succesiuni de probleme liniare, asociate proble-

Scanned with OKEN Scanner

mei date. Algoritmul mai este cunoscut sub numele de metoda planului de secțiune. Teoretic, succesiunea de probleme liniare asociate este infinită, dar practic, ca la orice metodă de aproximații succesive, ea se termină după un număr finit de pași. O descriere a acestei metode se găsește în [4].

2.5.3. Algoritmul lui Hildreth și D'Esopo

Am ales acest algoritm pentru a-l prezenta în detaliu, pentru că se pare că el este cel mai simplu algoritm de programat pentru un calculator electronic [4]. Algoritmul se aplică atunci cînd funcția obiectiv este strict convexă și constituie o metodă iterativă de rezolvare a problemei pătratice.

Fie problema $\mathcal{A}$:

$$Ax \leqslant b, \tag{24}$$

$$x \geqslant 0, \tag{25}$$

$$\min \left(p'x + \frac{1}{2}\, x'Cx \right), \tag{26}$$

unde $A(a_{ij})$ are m linii și n coloane, $b \in R^m$, $C(c_{ij})$ are n linii și n coloane, $p \in R^n$. Elementele a_{ij}, b_i și p_j sînt numere reale date, iar x_j sînt necunoscutele problemei.

Scriem problema duală $\mathcal{A}^*$ a lui $\mathcal{A}$ (vezi [4]), care constă în a găsi vectorii $x \in R^n$ și $u \in R^m$, astfel ca

$$Cx + A'u = -p, \tag{27}$$

$$u \geqslant 0, \tag{28}$$

$$\min \left(b'u + \frac{1}{2}\, x'Cx \right). \tag{29}$$

Din (27) deducem că $x = -C^{-1}(p + A'u)$, unde se vede că, pentru a cunoaște pe x, trebuie să cunoaștem pe u, adică x este funcție de u. Înlocuind pe x cu valoarea de mai sus în $\mathcal{A}^*$, problema duală capătă forma

$$u \geqslant 0, \tag{30}$$

$$\min \left(d'u + \frac{1}{2}\, u'Du \right), \tag{31}$$

Scanned with OKEN Scanner

unde prin d și D am notat matricele

$$d = b + AC^{-1}p\,; \quad D = AC^{-1}A'. \tag{32}$$

Găsirea lui u se face prin construirea în mod iterativ a unui șir $u^s \geqslant 0$, $s = 0,1,2,\ldots$. Se demonstrează că atunci cînd $s \to \infty$, șirul $x(u^s) \to \bar{x}$, unde prin $\bar{x}$ am notat soluția optimă a lui $\mathcal{A}$. Apar întrebările: care este u^0, cum se găsesc termenii următori ai șirului u^s și cînd se oprește procesul iterativ. Răspunsul este dat de următoarele:

a. vectorul $u^0 = 0 \in R^m$;

b. presupunînd că a fost construit termenul u^s, componentele iterației următoare u^{s+1} se calculează prin:

$$u_i^{s+1} = \max\left(0,\ v_i^{s+1}\right), \tag{33}$$

unde

$$v_i^{s+1} = -\frac{1}{d_{ii}}\left(d_i + \sum_{k=1}^{i-1} d_{ik}u_k^{s+1} + \sum_{k=i+1}^{m} d_{ik}u_k^s\right). \tag{34}$$

Elementele care intră în calculul lui v_i^{s+1} se văd în tabelul 15.

TABELUL 15

Iterație	u_1	u_2	...	u_{i-1}	u_i	u_{i+1}	...	u_m
0	$u_1^0 = 0$	$u_2^0 = 0$	....	$u_{i-1}^0 = 0$	$u_i^0 = 0$	$u_{i+1}^0 = 0$	..	$u_m^0 = 0$
. . . . .	. . . . .							
s	u_1^s	u_2^s	....	u_{i-1}^s	u_i^s	u_{i+1}^s	....	u_m^s
$s+1$	u_1^{s+1}	u_2^{s+1}	....	u_{i-1}^{s+1}	u_i^{s+1}			

c. Procesul iterativ se oprește atunci cînd se constată că $x(u^s)$ nu se mai modifică, sau cînd u nu se mai modifică.

Scanned with OKEN Scanner

Algoritmul lui Hilbreth și D'Esopo constă în următorii pași:

1. Se calculează matricea inversă C^{-1} a matricei C. (Acest algoritm se aplică numai cînd C este inversabilă).

2. Se calculează elementele d_i și d_{ij}, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant m$, ale lui d și D.

3. Se pornește cu $u^0 = 0$ și se organizează calculele ca în tabelul 16:

TABELUL 16

d	D						
d_1	d_{11}	d_{12}	...	$d_{1,i-1}$	d_{1i}	$d_{1,i+1}$ ...	d_{1m}
...	...						
d_m	d_{m1}	d_{m2}	...	$d_{m,i-1}$	d_{mi}	$d_{m,i+1}$ ...	d_{mm}
Iterația	u_1	u_2	...	u_{i-1}	u_i	u_{i+1} ...	u_m
0	0	0	...	0	0	0 ...	0
1	u_1^1	u_2^1	...	u_{i-1}^1	u_i^1	u_{i+1}^1 ...	u_m^1
...	...						
s	u_1^s	u_2^s	...	u_{i-1}^s	u_i^s	u_{i+1}^s ...	u_m^s
$s+1$	u_1^{s+1}	u_2^{s+1}	...	u_{i-1}^{s+1}	$\boxed{u_i^{s+1}}$		
...	...						

Calculul iterateratelor se continuă conform cu c. Erorile de calcul, eventual de rotunjire, nu sînt prea grave, ele ducînd cel mult la lungirea procesului iterativ. Algoritmul este relativ ușor programabil pentru un calculator, mai ales dacă se dispune de un program de bibliotecă de găsire a inversei C^{-1}.

Scanned with OKEN Scanner

2.6. REZOLVAREA NUMERICĂ A ECUAȚIILOR DIFERENȚIALE ORDINARE

În cele ce urmează vom prezenta îmbunătățirea treptată a cîtorva metode numerice de rezolvare a ecuațiilor diferențiale ordinare de ordinul întîi. Aceste ecuații sînt legături funcționale între variabila independentă x, variabila dependentă y și derivata dy/dx, adică

$$F(x,y,y') = 0, \tag{35}$$

unde prin y' am notat $y' = dy/dx$.
Presupunem că ecuația (35) poate fi rezolvată în raport cu y', adică

$$y' = f(x,y). \tag{36}$$

A rezolva ecuația (36), înseamnă a determina funcția y care o satisface. Această funcție depinde însă de un parametru c, care este o constantă de integrare, adică soluția y este de forma $y = g(x, c)$. Ea reprezintă deci o familie de curbe în planul xy, numit curbe integrale. De exemplu, $y' = \cos x$ conduce la $y = \sin x + c$, care reprezintă o familie de curbe ca cele din figura 13. Dintre aceste curbe, cel mai adesea ne interesează numai una și anume aceea care trece printr-un punct dat M, adică acea curbă care îndeplinește condiția

$$y_0 = g(x_0, c), \tag{37}$$

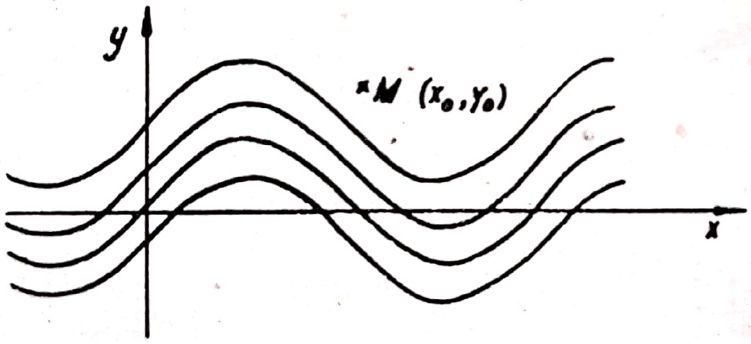


Fig. 13. Familia curbelor integrale

unde x_0 și y_0 sînt valori date, numite *valori inițiale*. Deci, cunoscînd valorile inițiale, se determină c, astfel încît curba integrală să treacă prin M. De exemplu, $x_0 = 0$, $y_0 = 2$ implică $c = 2$ și deci curba integrală căutată este $y = \sin x + 2$.

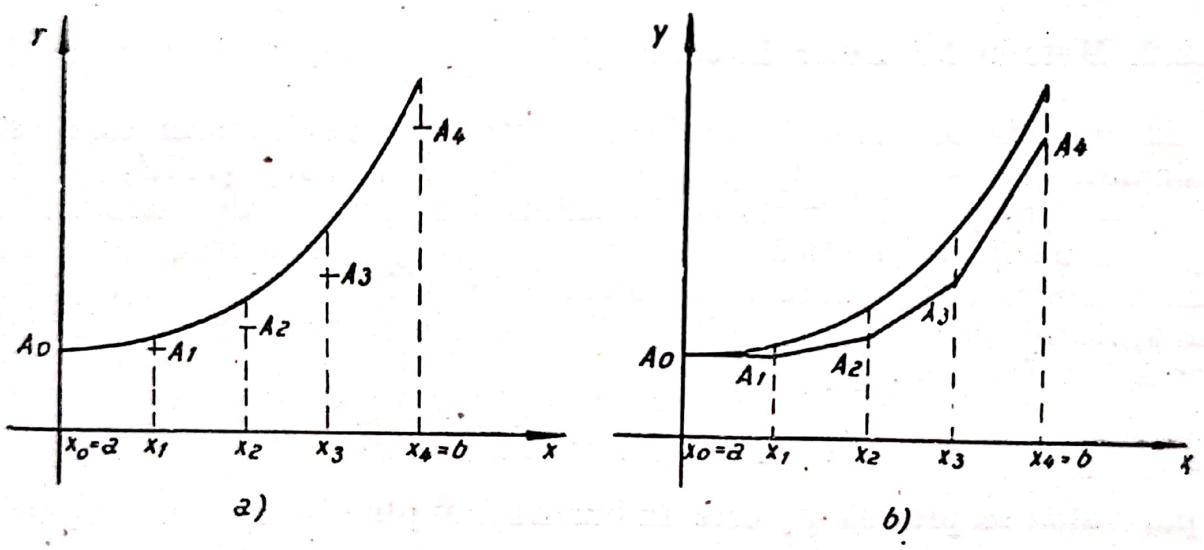
Găsirea soluției $y = g(x,c)$, care să satisfacă anumite condiții inițiale date, este adeseori foarte dificilă. De aceea se caută curba y prin

Scanned with OKEN Scanner

puncte discrete, în număr oricît de mare, dar totuşi finit. După ce punctele discrete au fost găsite, se unesc prin segmente de dreaptă şi deci se obţine o aproximare a soluţiei exacte (necunoscute) printr-o linie frîntă. Această operaţie formează *rezolvarea numerică a ecuaţiei diferenţiale cu valorile iniţiale date*. Există numeroase metode de rezolvare numerică. Vom începe prin prezentarea celei mai simple şi anume :

2.6.1. Metoda lui Euler

Fie de rezolvat *numeric* ecuaţia $y' = f(x,y)$, cînd x ia valori în intervalul $[a, b]$ şi astfel încît $y(x_0) = y_0$, cu x_0 şi y_0 daţi. Fiind vorba de puncte discrete ale curbei integrale, ele corespund la valori discrete ale lui x. Vom împărţi segmentul $[a, b]$ în N părţi egale prin punctele $x_0 = a, x_1, \ldots, x_N = b$, adică $x_{i+1} = x_i + h$, $i = 0,1,\ldots, N-1$, unde h este pasul cu care a fost împărţit segmentul. Metoda lui Euler



a) Urma soluţiilor numerice
b) Soluţia aproximativă sub forma unei linii frînte

Fig. 14.

indică modul în care se găsesc valorile $y_1, \ldots, y_N$ corespunzătoare lui $x_1, \ldots, x_N$ (x_0 şi y_0 sînt cunoscute, ca valori iniţiale). Punctele $A_i(x_i, y_i)$, $i = 0,1,\ldots, N$ definesc „urma" curbei care aproximează soluţia ecuaţiei date. Presupunînd că se cunoaşte soluţia exactă, adică curba integrală exactă, geometric, metoda lui Euler se ilustrează ca în figura 14, *a,b* :

Scanned with OKEN Scanner

A rămas să mai arătăm cum se calculează y_i. Se folosește formula tangentei într-un punct (x_0,y_0) la o curbă $p(x)$:

$$y - y_0 = p'(x_0)(x - x_0).$$

În cazul rezolvării ecuației diferențiale, curba este $y(x)$ și deci $y'(x_0) = f(x_0,y_0)$. Atunci obținem

$$\begin{aligned} y_1 &= y_0 + (x_1 - x_0)f(x_0,y_0) \\ y_2 &= y_1 + (x_2 - x_1)f(x_1,y_1) \\ &\cdots\cdots\cdots\cdots \\ y_{i+1} &= y_i + (x_{i+1} - x_i)\, f(x_i,y_i),\ i = 0,1,\ldots,\ N-1. \end{aligned} \tag{38}$$

Soluția aproximativă aproximează cu atît mai bine soluția exactă cu cît pasul h este mai mic. Totuși, oricît de mic ar fi h, metoda lui Euler dă erori mari, care cresc rapid.

2.6.2. Metoda lui Euler îmbunătățită

În metoda anterioară erorile se datorează modului în care este calculată panta fiecărui segment din linia frîntă $A_0A_1\ldots A_N$, anume $y'(x_i) = f(x_i, y_i)$. În metoda îmbunătățită, panta, de exemplu în x_1, este media aritmetică a lui $f(x_0,y_0)$ și $f(x_1,y_1)$. Valorile x_0, y_0 și x_1 sînt cunoscute. Dar care este valoarea lui y_1? Aceasta se calculează cu ajutorul primei relații (38), adică prin metoda lui Euler. Notăm această valoare cu p_1:

$$p_1 = y_0 + (x_1 - x_0)\, f(x_0,y_0).$$

Apoi valoarea prezisă p_1 este îmbunătățită pînă la valoarea y_1 prin formula

$$y_1 = y_0 + \frac{1}{2}(x_1 - x_0)\ [f(x_0,y_0) + f(x_1,p_1)].$$

Valoarea y_2 corespunzătoare lui x_2 este prezisă prin p_2.

$$p_2 = p_1 + (x_2 - x_1)f(x_1, p_1),$$

apoi p_2 este îmbunătățită pînă la y_2 prin

$$y_2 = y_1 + \frac{1}{2}(x_2 - x_1)[f(x_1,y_1) + f(x_2,p_2)]$$

Scanned with OKEN Scanner

Scanned with OKEN Scanner

și deci

(39)

a. $$p_{i+1} = p_i + (x_{i+1} - x_i) f(x_i, p_i);$$

b. $$y_{i+1} = y_i + \frac{1}{2}(x_{i+1} - x_i)[f(x_i, y_i) + f(x_{i+1}, p_{i+1})]$$

pentru $i = 2, \ldots, N - 1$.

Prima variantă a acestei metode este aceea în care p_1 și y_1 se calculează ca mai sus, dar pentru calculul lui p_2 se folosește valoarea y_1 (nu p_1), adică în general

(40) $$p_{i+1} = y_i + (x_{i+1} - x_i) f(x_i, y_i), \quad i, \ldots, N - 1.$$

A doua variantă constă în iterarea calculării fiecărei valori y_{i+1}, $i = 0, \ldots, N - 1$ pentru x_i, x_{i+1} și p_{i+1} neschimbați. Aceasta se face astfel: se calculează prima valoare y_{i+1} (după 39) și fie ea y_{i+1}^1. Apoi în locul lui y_i se pune y_{i+1}^1 și se calculează cu (39). b valoarea y_{i+1}^2; apoi y_{i+1}^2 ia locul lui y_{i+1}^1 și se continuă procesul pînă cînd

(41) $$|y_{i+1}^k - y_{i+1}^{k+1}| < \varepsilon,$$

unde $\varepsilon > 0$ este un număr dat arbitrar.

Evident, această variantă se poate aplica și primei variante de mai sus, constituind deci varianta a treia.

Procedeul de prezicere și apoi de îmbunătățire a valorii prezise este unul din cele mai eficiente procedee de rezolvare a problemelor cu valori inițiale.

2.6.3. Metoda de anticipare — corectare a lui Euler

Valorile x_0, y_0 sînt cunoscute ca valori inițiale. Valoarea y_1 se calculează cu ajutorul metodei lui Euler îmbunătățită (sau a unei variante a ei). Valoarea y_2 este anticipată (prezisă) prin

(42) $$p_2 = y_0 + 2(x_2 - x_1) f(x_1, y_1),$$

apoi ea este corectată prin

(43) $$y_2 = y_1 + \frac{1}{2}(x_2 - x_1)[f(x_1, y_1) + f(x_2, p_2)]$$

și deci

(44) a. $$p_{i+1} = y_{i-1} + 2(x_{i+1} - x_i)f(x_i, y_i)\ ;$$

b. $$y_{i+1} = y_i + \frac{1}{2}(x_{i+1} - x_i)[f(x_i, y_i) + f(x_{i+1}, p_{i+1})]$$

pentru $i = 2, \ldots, N-1$.

2.6.4. Metoda de anticipare — corectare îmbunătățită a lui Euler

Valorile x_0, y_0 sînt cunoscute ca valori inițiale. Valoarea y_1 se calculează cu ajutorul metodei lui Euler îmbunătățită (sau a unei variante a ei). Valoarea y_2 este anticipată (prezisă) prin p_2 după (42), apoi este corectată prin (43). Notăm această corecție prin q_2. Valoarea corectată q_2 este îmbunătățită pînă la y_2 prin

(45) $$y_2 = \frac{p_2 + 4q_2}{5}$$

și deci

(46) a. $$p_{i+1} = y_{i-1} + 2(x_{i+1} - x_i)f(x_i, y_i)\ ;$$

b. $$q_{i+1} = y_i + \frac{1}{2}(x_{i+1} - x_i)[f(x_i, y_i) + f(x_{i+1}, p_{i+1})]\ ;$$

c. $$y_{i+1} = \frac{p_{i+1} + 4q_{i+1}}{5}$$

pentru $i = 2, \ldots, N-1$.

Dintre metodele lui Euler, această metodă dă cele mai bune rezultate.

2.6.5. Metoda de anticipare — corectare a lui Adams și Bashforth

Valorile x_0, y_0 sînt cunoscute ca valori inițiale. Valorile y_1, y_2 și y_3 corespunzătoare lui x_1, x_2 și x_3 se calculează folosind una din metodele anterioare. Cu ajutorul acestora anticipăm valoarea lui y_4 prin

(47) $$p_4 = y_3 + \frac{1}{24}(x_4 - x_3)[55f_3 - 59f_2 + 37f_1 - 9f_0],$$

unde $f_i = f(x_i, y_i)$, $i = 0,1,2,3$. Apoi calculăm valoarea $f_4 = f(x_4, p_4)$ cu care corectăm pe p_4 pînă la y_4 prin

(48) $$y_4 = y_3 + \frac{1}{24}(x_4 - x_3)[f_1 - 5f_2 + 19f_3 + 9f_4]$$

Scanned with OKEN Scanner

și deci

$$(49)\quad \text{a. } p_{i+1} = y_i + \frac{1}{24}(x_{i+1} - x_i)[55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}];$$

$$\text{b. } y_{i+1} = y_i + \frac{1}{24}(x_{i+1} - x_i)[f_{i-2} - 5f_{i-1} + 19f_i + 9f_{i+1}]$$

pentru $i = 4, \ldots, N-1$, unde $f_k = f(x_k, y_k)$ pentru $k = i-3$, $i-2$, $i-1$, i iar $f_{i+1} = f(x_{i+1}, p_{i+1})$. Pentru calcularea lui p_5 sînt necesare f_1, f_2, f_3, deja calculate, și f_4, dar nu valoarea calculată anterior, ci valoarea recalculată cu ajutorul lui y_4, adică $f_4 = f(x_4, y_4)$. Apoi se aplică (49 a) și (49 b).

2.6.6. Metoda de anticipare — corectare a lui Adams — Bashforth îmbunătățită

Modificarea constă în modul în care se calculează y_i, $i = 4, \ldots, N-1$. Notăm y_4, calculat în (48), cu q_4. Îmbunătățim pe q_4 pînă la valoarea y_4 prin

$$(50)\qquad y_4 = \frac{19p_4 + 251q_4}{270}.$$

Analog, notăm y_{i+1}, dat de (49a), cu q_{i+1} și deci

$$(51)\qquad y_{i+1} = \frac{19p_{i+1} + 251\,q_{i+1}}{270}$$

pentru $i = 4, \ldots, N-1$, unde $f_k = f(x_k, y_k)$ pentru $k = i-3$, $i-2$, $i-1$, i, iar $f_{i+1} = f(x_{i+1}, p_{i+1})$.

2.6.7. Metoda lui Runge-Kutta

Metoda pe care o vom descrie mai jos face parte din clasa de metode Runge-Kutta. Aceste metode sînt analoge ca concepție cu metodele lui Euler, fiind bazate pe calculul pantei segmentului de dreaptă prin care se aproximează pe distanța h curba integrală exactă. Metoda de mai jos se folosește fie ca metodă de sine stătătoare pentru găsirea soluției numerice, fie ca metodă de calculare a valorilor inițiale, necesare în metodele de anticipare — corectare.

Scanned with OKEN Scanner

Notăm cu $h = x_{i+1} - x_i$, $i = 0,1,\ldots, N-1$. Valorile x_0, y_0 și h sînt cunoscute. Calcularea valorilor y_i, $i = 1,\ldots, N-1$ se face după formulele:

(51) a. $$p_{i+1} = hf(x_i, y_i);$$

b. $$q_{i+1} = hf\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}p_{i+1}\right);$$

c. $$r_{i+1} = hf\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}q_{i+1}\right);$$

d. $$s_{i+1} = hf(x_i + h, y_i + r_{i+1});$$

e. $$y_{i+1} = y_j + \frac{1}{6}[p_{i+1} + 2q_{i+1} + 2r_{i+1} + s_{i+1}]$$

pentru $i = 0,1,\ldots, N-1$. Valoarea y_{i+1} este media ponderată a valorilor calculate anterior.

Această metodă dă rezultate la fel de bune ca și metoda de la § 2.6.6. Un dezavantaj al acestei metode este acela că necesită calcularea valorilor lui f pentru patru perechi diferite de valori ale lui x și y, pentru fiecare iterație.

Pentru detalii privind metodele descrise se poate cosulta [2].

BIBLIOGRAFIE

[1] Beale, E. M. L. — *On Quadratic Programming*, Naval Research Logistics Quaterly, vol. 6, 1969.

[2] Beckett, R., Hurt, J. — *Numerical Calculation and Algorithms*, McGrow — Hill Book Company, New-York, 1967.

[3] Churchman, W., Arckoff, R. L., Arnoff, E. L. — *Introduction to Operations Research*, New-York, 1957.

[4] Dragomirescu, M., Malița, M. — *Programarea matematică*, Ed. Științifică, București, 1968.

[5] Elmagharby, S. E. — *Proiectarea sistemelor de producție*, traducere, Ed. Tehnică, București, 1968.

[6] Hadley, G. — *Nonlinear and Dynamic Programming*, Addison-Wesley Publishing Co., London, 1964.

[7] IBM System/360 — *Fortran IV Language*, C28—6515—7.

Scanned with OKEN Scanner

[8] Marcus, S. — *Noțiuni de analiză matematică*, Eneiclopedia de buzunar, Ed. Științifică, București, 1967.

[9] Künzi, H. P., Tzschach, H. G., Zender., C. A. — *Numerical Methods of Mathematical Optimization*, Acad Press, New-York, London, 1968.

[10] Savas, E. S. — *Conducerea cu calculatoare a proceselor industriale*, traducere, Ed. Tehnică, București, 1969.

[11] Simonard, M. — *Programation linéaire*, Dunod, Paris, 1962.

[12] Văduva, I. — *On a Mathematical Model Concerning Endemy of Tuberculosis*, in Internationale Tagung Über Mathematische Statistik und Ihre Anwendungen, Berlin, vom 6. bis 9. Mai 1966, pp. 183—188, Akademie — Verlag, Berlin

[13] Wolfe, P. — *The Simplex Method for Quadratic Programming*, Econometrica, Nr. 27, 1959.

Scanned with OKEN Scanner

Capitolul III

INTRODUCERE ÎN LIMBAJUL FORTRAN

3.1. CARACTERISTICI ALE LIMBAJELOR DE PROGRAMARE ORIENTATE PENTRU CERCETAREA ŞTIINŢIFICĂ

După cum din punct de vedere al rezolvării cu calculatorul distingem două clase mari de probleme — prelucrări de date şi probleme ştiinţifice —, tot aşa şi limbajele de programare se împart în două mari categorii: limbaje pentru prelucrarea datelor şi limbaje ştiinţifice. În ultimul timp clasificarea limbajelor s-a adîncit astfel încît se poate vorbi de mult mai multe tipuri de limbaje; aceasta deoarece categoriile de aplicaţii ale calculatoarelor s-au diversificat.

Ne vom opri puţin asupra limbajelor de programare orientate pentru cercetarea ştiinţifică. Problemele de cercetare ştiinţifică, ca probleme de calcul, se caracterizează în primul rînd prin aceea că folosesc puţine date de intrare, necesită un timp îndelungat de calcul şi utilizează algoritmi laborioşi care conduc la rezultate de calcul (ieşiri) relativ puţine. Prin urmare, problemele de cercetare ştiinţifică presupun existenţa unor modele matematice care nu întotdeauna pot fi rezolvate analitic; aceste modele se rezolvă de regulă prin metode numerice. De aceea, limbajele de programare orientate pentru cercetarea ştiinţifică trebuie să satisfacă în principal următoarele cerinţe:

a) să dispună de macro-instrucţiuni pentru efectuarea principalelor operaţii aritmetice şi logice (adunare, scădere, înmulţire, împărţire, ridicare la putere, extragerea rădăcinii pătrate etc.); sau pentru calculul valorilor unor expresii aritmetice şi logice;

b) să permită calculul funcţiilor elementare (e^x, ln x, sin x, cos x, tg x, etc.);

c) să permită introducerea automată a datelor în calculator şi scoaterea automată a rezultatelor.

Printre limbajele de programare orientate pentru cercetarea ştiinţifică, cele mai răspîndite sînt FORTRAN, ALGOL şi PL/I. Dintre acestea ALGOL şi PL/I sînt cele mai evoluate, ele permiţînd programarea cu uşurinţă a tuturor problemelor de cercetare. Limbajul FORTRAN, deşi mai sărac în astfel de facilităţi, permite totuşi programarea majorităţii problemelor de cercetare ştiinţifică întîlnite. FORTRAN, prescurtare de la FORmula TRANslator = translator de formule (lb. engleză), a fost definit în anul 1957 pentru calcula-

Scanned with OKEN Scanner

toarele de tip IBM. Apoi el s-a dezvoltat pînă la forma limbajului FORTRAN II, ca ulterior să se extindă la versiunea cea mai completă, FORTRAN IV, pentru care există compilatoare la majoritatea calculatoarelor moderne. Cu ajutorul limbajului FORTRAN IV se pot programa probleme de matematică precum și orice alte probleme cu aplicații științifice și se pot prelucra fișiere bogate de date, în special numerice.

Scopul acestui capitol este, în primul rînd, de a introduce pe cititor în limbajul de programare FORTRAN, cel mai popular limbaj cunoscut aproape de toți utilizatorii sistemelor de calcul existente. Se vor prezenta numai o parte din instrucțiunile limbajului, care permit oricărui cercetător să rezolve cu ușurință problemele sale, urmînd ca cititorul interesat să-și perfecționeze cunoștințele printr-o activitate practică susținută și prin utilizarea manualelor de care dispun diversele sisteme de calcul existente.

3.2. DESCRIEREA CARTELEI ȘI MULȚIMEA DE CARACTERE

Limbajul FORTRAN este conceput în mod special pentru introducerea informațiilor în calculator prin intermediul cartelei perforate; acest fapt va influența mult forma limbajului.

O cartelă are 80 de coloane, cu posibilitatea de a realiza perforații în 12 poziții diferite pe fiecare coloană.

Un program FORTRAN apare ca o succesiune de instrucțiuni, o instrucțiune fiind de regulă scrisă pe o singură linie, deci perforată pe o cartelă. Instrucțiunile sînt scrise pe formulare speciale, fiind liniate pe verticală tot cu 80 coloane. Există o corespondență biunivocă între coloanele formularului și cele ale cartelei; o linie pe formular corespunde unei cartele. În cele ce urmează ne vom referi numai la cartelă, subînțelegînd că cele ce se vor preciza sînt valabile și pentru ceea ce se scrie pe o linie a formularului. Vom descrie forma de prezentare a instrucțiunilor limbajului FORTRAN pe cartelă (vezi fig. 2).

Coloana 1 în mod normal nu conține nimic. Ea poate conține litera *C* (literă mare), ceea ce înseamnă că tot conținutul acelei cartele este un comentariu care-l privește pe programator sau pe cel ce va utiliza programul și acest comentariu va fi ignorat de către calculator; el însă va apare listat la imprimantă odată cu celelalte instrucțiuni ale programului. Coloana 1 poate să conțină semne speciale care au sens pentru sistemul de operare utilizat.

Scanned with OKEN Scanner

Coloanele 2—5 pot conține un *număr de instrucțiune*, care are rol de *etichetă*, în sensul că alte instrucțiuni ale programului se pot referi la instrucțiunea care poartă acest număr. Blancurile (spațiile libere) sau cifrele zero nesemnificative (din fața altor cifre diferite de zero) sînt ignorate de către compilator.

Coloana 6 nu conține în mod normal nimic, dar dacă ea conține orice caracter (diferit de zero sau blanc), atunci cartela corespunzătoare conține o *continuare* a instrucțiunii de pe cartela precedentă. De obicei cartelele care conțin continuări sînt numerotate cu 1, 2,..., în cazul cînd instrucțiunea pe care ele o reprezintă este foarte lungă. Numărul de cartele-continuări este limitat (la majoritatea versiunilor FORTRAN el este de cel mult 19).

Coloanele 7—72 conțin o instrucțiune FORTRAN, ca de exemplu

$$X = Y + A * W$$

READ (1,1) *A*, *B*

Blancurile sînt în general ignorate, exceptînd cazul unor anumite instrucțiuni FORMAT. Deci toate instrucțiunile FORTRAN se scriu în coloanele 7—72 inclusiv.

Coloanele 73—80 nu sînt folosite în limbajul FORTRAN. Ele pot fi însă folosite de către programator pentru numerotarea cartelelor, cu scopul de a restabili cu ușurință ordinea cartelelor dintr-un pachet în cazul în care s-ar amesteca din eroare cartelele.

Caracterele (semnele sintactice) pe care le folosește limbajul FORTRAN sînt:

Cele 26 litere mari ale alfabetului latin

A B C D E F G H I J K L M N O P Q R
S T U V W X Y Z

Cele 10 cifre arabe: 1 2 3 4 5 6 7 8 9 0.

Cele 10 simboluri: + — * / = . , ( ) ' (apostroful).

Pentru a se face distincție între litera *O* și cifra zero se convine ca aceasta din urmă să fie notată Ø.

Nu există caracter special pentru „spațiu", deoarece un spațiu liber se obține prin lăsarea unei coloane libere (blanc). Legat de sistemul de operare și de felul de calculator utilizat, în mulțimea de caractere există și altfel de caractere, ca de exemplu $ (semnul pentru „dolar") și £ (semnul pentru „lira sterlină").

Scanned with OKEN Scanner

3.3. CONSTANTE ŞI VARIABILE: IDENTIFICATORI ŞI DECLARĂRI DE TIPURI

Distincţia care se face între o *constantă* şi o *variabilă* provine din semnificaţia matematică a acestor noţiuni.

Constantele pot fi numerice, logice (*LOGICAL*) şi literale. Constantele numerice pot fi *întregi* (*INTEGER*), *reale* (*REAL*) şi complexe (*COMPLEX*); constantele logice pot lua una din valorile ·*TRUE*· (= adevărat) sau ·*FALSE*· (= fals), iar o constantă literală poate fi orice şir de caractere.

Constantele întregi se scriu în mod obişnuit; cifrele zero la începutul numărului nu sînt semnificative. Depinzînd de calculatorul utilizat, fiecare compilator FORTRAN presupune existenţa unor întregi într-un interval de forma $(-2^b, 2^b - 1)$. De exemplu, pentru calculatoarele IBM/360 şi FELIX C—256, $b = 31$, deci se pot folosi numere întregi în intervalul (2 147 483 648, 2 147 483 647); numere întregi în afara acestui interval nu pot fi utilizate într-un program FORTRAN.

Un număr întreg poate fi însoţit de un semn + sau —; cînd semul lipseşte, se presupune că este +.

O constantă reală se scrie ca un şir finit de cifre zecimale care au între ele un *punct* zecimal (În notaţia anglosaxonă, pentru numerele zecimale se foloseşte punctul în loc de virgulă).

O altă scriere a constantei reale este aceea care utilizează un exponent zecimal sub forma *R E I*, unde *R* este o *mantisă* (număr scris ca mai sus), *E* este un simbol (care reprezintă pe 10 ca bază a unei puteri), iar *I* este un întreg de două cifre cu semn.

În mod obişnuit, un anumit calculator rezervă pentru fiecare număr întreg sau real un anumit număr de locaţii de memorie (de ex. baiţi la IBM/360 şi FELIX C—256). Dacă în anumite calcule este nevoie să se folosească numere întregi sau reale mai mari (de *lungime dublă*), anumite compilatoare FORTRAN permit — prin folosirea unor instrucţiuni adecvate — alocarea unui număr de *locaţii* de memorie, dublu faţă de cel standard. Scrierea constantelor reale în dublă precizie utilizează litera *D* în locul literei *E*.

O constantă *complexă* este o pereche ordonată de constante reale, scrise în paranteze şi despărţite prin virgulă; ambele numere reale (partea reală şi partea imaginară) trebuie să aibă aceeaşi precizie.

O constantă literală poate fi scrisă în două moduri şi anume:

a) şirul de semne alfanumerice este inclus între apostrofuri;

b) şirul este precedat de *nH*, unde *n* este numărul de caractere din şir (inclusiv spaţiile).

Scanned with OKEN Scanner

Exemple de constante :
173Ø1 752 (constante întregi) ;
153.Ø2 75.41Ø.ØØ2 (constante reale) ;
Ø.573*E*—Ø2 Ø.9134*E* + 1Ø (constante reale) ;
Ø.5313*D*—Ø2 Ø.5931 *D*—71 (constante reale duble precizie) ;
.*TRUE*. .*FALSE*. (constante logice ; atenţie la scriere) ;
(Ø.5*E* + 21, Ø.732*E*—Ø2) (constantă complexă) ;
'*INTRARE*' 4*H APA* (constante literale).

Aşa cum am precizat mai înainte, fiecărei constante i se alocă o anumită memorie în funţie de tipul constantei. La sistemul de calcul IBM 360 sau FELIX C—256 alocarea este următoarea :

întreg 4 baiţi real (dublă precizie) 8 baiţi
real 4 baiţi logic 4 baiţi
literal 1 bait pentru fiecare caracter.

Orice *variabilă* care apare într-un program FORTRAN poartă un nume (are un *identificator*), iar fiecarei nume i se alocă o locaţie din memoria calculatorului, unde se memorează o valoare intermediară sau finală a variabilei. Deci, într-un anume sens, identificatorul poate fi considerat *adresa* locaţiei de memorie unde se memorează variabila.

Un *identificaror FORTRAN* începe întotdeauna cu o literă care poate fi urmată numai de *litere* sau *cifre*, astfel încît numărul total de caractere să fie cel mult egal cu 6*. Este util ca identificatorii să fie sugeraţi de context, însă ei pot fi aleşi şi după dorinţa programatorului. De exemplu, *SUPR* poate să desemneze o arie, *AB* poate să desemneze produsul a două numere (fără să însemne însă *A* înmulţit cu *B*).

Variabilele FORTRAN pot fi de mai multe tipuri : întregi, reale şi logice. Distincţia între variabile care reprezintă numere întregi şi cele care reprezintă numere reale se face fie în mod *implicit*, după forma sintactică a identificatorului, fie prin declararea *explicită* a tipului. Astfel, un identificator care începe cu una din literele *I*, *J*, *K*, *L*, *M*, *N* şi nu a fost declarat explicit ca fiind de alt tip, reprezintă o variabilă întreagă ; dacă identificatorul începe cu oricare altă literă, atunci el reprezintă o variabilă reală. De exemplu, numele

XYZRAZAGREUTVOLUM

reprezintă variabile reale, în timp ce

IJMINCARELUNGMARE

reprezintă variabile întregi.

* Aceasta este situaţia la sistemele IBM 360. Pentru alte sisteme de calcul lungimea identificatorului este diferită de 6 (de ex. la IBM 1130 lungimea identificatorului este 5, la FELIX C—256 lungimea este 8).

Scanned with OKEN Scanner

Declararea explicită a tipului se face prin instrucţiuni de forma: *Tipul listă.*

Primul element al declarării desemnează tipul identificatorului în limba engleză, după care urmează o listă de nume despărţite prin virgulă. Tipul este după caz *REAL* (număr real), *INTEGER* (număr întreg), *COMPLEX* (variabilă complexă), *LOGICAL* (variabilă logică). O variabilă logică este o variabilă nenumerică şi ia valorile ·*TRUE*· (adevărat) sau ·*FALSE*· (fals). (A se remarca forma sintactică a celor două valori ale unei variabile logice). De exemplu,

REAL INT, MAX, LAM

declară cele trei variabile ca fiind reale, în timp ce

INTEGER AM, VAR, DISP

declară variabilele din listă ca fiind întregi. Instrucţiunea de declarare

LOGICAL LUNG, MIC, AX

precizează că cele trei variabile nu sînt numere, ci sînt variabile logice care vor apare în programul *FORTRAN*.

Alocarea memoriei pentru variabile se face în acelaşi mod ca pentru constante.

În calcule ştiinţifice intervin uneori variabile care sînt *liste* unidimensionale sau multidimensionale de alte variabile (de ex. vectori, matrice, etc.), deci variabile cu indici (subscripţi). Pentru astfel de variabile se pot folosi identificatori de forma *X*1 (pentru a nota X_1), *A*13 (pentru a desemna elementul a_{13} al unei matrice), dar există o metodă mai puternică de referire la astfel de variabile care este dată de noţiunea de *masiv*. Orice masiv desemnează o listă de variabile, care au o poziţie relativă bine precizată una faţă de alta. Masivul poartă un nume. Un element al masivului se identifică prin indicele pe care-l poartă. Declararea unui masiv este de forma

DIMENSION listă,

unde lista conţine nume de masive, fiecare nume avînd în paranteze numărul maxim de elemente al masivului. Cînd masivul are mai mulţi subscripţi, atunci dimensiunile maxime sînt *separate* prin virgulă. De exemplu, instrucţiunea

DIMENSION A(2Ø), *X*(1Ø, 1Ø)

declară vectorul *A* de 2Ø componente şi matricea *X* ca avînd 1Ø linii şi 1Ø coloane. Dacă pe parcursul programului este nevoie să ne referim

Scanned with OKEN Scanner

la elementul a_7, atunci vom scrie $A(7)$. În exemplul anterior este permisă şi referirea sub forma $X(I, J)$, unde I şi J sînt întregi, $1 \leqslant I, J \leqslant 1Ø$.

În concluzie, indicii unui masiv pot fi numai numere întregi pozitive sau variabile întregi care iau numai valori pozitive. Un masiv în FORTRAN poate avea cel mult şapte subscripţi.

Memorarea elementelor unui masiv se face astfel. Componentele unui vector se memorează în ordinea crescătoare a indicelui; componentele unui masiv cu mai mult de un indice se memorează în ordinea crescătoare a indicilor, astfel încît indicele din stînga creşte cel mai rapid, iar indicii următori cresc din ce în ce mai încet. De exemplu, elementele matricei X, la care ne-am referit anterior, se memorează în ordinea

$$X(1,1) \quad X(2,1) \ldots X(1Ø,1) \quad X(1,2) \quad X(2,2) \ldots$$
$$\ldots X(1Ø,2) \ldots\ldots X(1Ø,1Ø),$$

adică se memorează ca un vector format prin alăturarea consecutivă a coloanelor matricei (memorarea pe coloane).

Cînd masivul este de un anumit tip, atunci declararea poate începe cu tipul (omiţîndu-se specificarea *DIMENSION*). De exemplu, declararea

$$INTEGER \quad DISP(2Ø), \; AX(15,15)$$

înlocuieşte declarările

$$DIMENSION \quad DISP(2Ø), \; AX(15,15)$$

$$INTEGER \quad DISP(2Ø), \quad AX(15,15)$$

Desigur sînt valabile şi declarări combinate, de forma

$$INTEGER \quad X, \; Y(15), \quad Z(25,5)$$

în care unii sau mai mulţi identificatori (în cazul nostru X) desemnează o variabilă.

Dacă într-un program avem nevoie să utilizăm variabile cu dublă precizie, atunci ele trebuie declarate. Această declarare este de forma

$$DOUBLE \quad PRECISION \quad D, \quad E, \quad F, \quad A(1Ø, \; 1Ø).$$

În final, precizăm că, într-un *program* sau *subprogram FORTRAN*, toate instrucţiunile de declarare se plasează la începutul programului.

Scanned with OKEN Scanner

Exerciții

1. Care din construcțiile sistactice următoare nu sînt identificatori FORTRAN valabili și de ce?

INX,1 *AM*/3 *AM*3 *I*355 3*IJJ*

*I*4567Ø0 *OM*Ø *XXXXX*1 *X*2*Y*3*Z*45

2. Ce greșeli sintactice conțin construcțiile sintactice următoare?

a) *DIMENSION*, I47(21,54,75,2)*X*(17,18)*Y*(3)

b) *INTEGER A*(7,8) *B*(5,2), *D*(2,3)

c) *LOGICAL L*(5.2), *LOG*(4,5)

INTEGER LOG (3,4)

d) *COMPLEX C*1 *C*2, *C*3

e) *DOUBLE PRECISION X*(2), *Y*(3)

3. Care din următoarele constante întregi nu sînt scrise corect?

213245 1547. −3459 ± 131

4. Care din următoarele constante reale sînt scrise incorect?

10. +31.52 −3,15 17*E*+Ø2 Ø.17Ø*E*−Ø3

5. Care din următoarele constante complexe nu au scriere acceptabilă?

(2,3.) (3.5, 7.5) (Ø.3*E*+1Ø, Ø.5*D*−Ø1)

6. Să se specifice tipul variabilelor reprezentate de următoarele nume

INTR, *LAC*, *TIMP*, *AMAX*, *NORA*, *XNORMA*

3.4. EXPRESII ARITMETICE ȘI LOGICE

3.4.1. Expresii aritmetrice

Cele mai simple espresii aritmetice sînt cele determinate de un singur element (expresii primare) și anume: constante, variabile, variabile cu indici (elemente ale unui masiv), funcții și expresii incluse între paranteze.

Scanned with OKEN Scanner

Expresiile aritmetice mai complicate se formează din două sau mai multe elemente cu ajutorul următoarelor operații:

ridicarea la putere, notată	**
înmulțirea, notată	*
împărțirea, notată	/
adunarea, notată	+
scăderea, notată	—

Ordinea de execuție a operațiilor aritmetice este cea obișnuită, adică: ridicarea la putere, înmulțirea și împărțirea, adunarea și scăderea. Dacă în calcule intervine și o funcție (vezi §3.5), atunci valoarea ei se determină înainte de efectuarea altor operații aritmetrice.

Cînd se construiesc expresii aritmetice, trebuie să se aibă în vedere faptul că nu sînt permiși doi operatori aritmetrici consecutivi: dacă o asemenea situație se întîlnește, atunci operatorii trebuie separați prin paranteze. Exemple de expresii aritmetice sînt următoarele:

$$A + B + C - D$$

$$A \ ** \ 2 + B \ ** \ 3 - C * D$$

$$A * (-B) + C \ ** \ (-D)$$

Înmulțirea și împărțirea se efectuează de la stînga la dreapta expresiei. De exemplu, expresia:

$$P * V / R * T$$

este echivalentă cu expresia:

$$P * V * T / R$$

Dacă nu aceasta a fost intenția programatorului, ci să dividă pe $P * V$ la $R * T$, atunci trebuie ca expresia să se scrie:

$$P * V / (R * T) \text{ sau } P * V/R/T.$$

Dacă există operații de ridicare la putere consecutive, atunci ele se execută de la dreapta la stînga, adică expresia:

$A \ ** \ B \ ** \ C$ este echivalentă cu $A \ ** \ (B \ ** \ C)$.

Tipul rezultatului operațiilor aritmetice se determină astfel: dacă I, R, D sînt, respectiv, tipurile întreg, real, dublă precizie, $I \subset R \subset D$, atunci rezultatul operației aritmetice este de același tip cînd

Scanned with OKEN Scanner

cei doi termeni sînt de același tip, și este de tipul termenului mai „tare'' cînd cei doi termeni au tipuri diferite*. De exemplu, expresiile:

$$I/J + L - M + LL * N$$

$$LM * LN - L + N$$

au tipul întreg, pe cînd expresiile:

$$L/M + X - Y$$

$$I + J - A + B$$

au tipul real.

Să observăm că, dacă într-o împărțire cele două elemente sînt de tipul întreg, atunci rezultatul este tot întreg și anume este partea dinainte de virgulă a cîtului celor doi întregi. De exemplu, dacă $I = 7$ și $J = 3$, atunci I/J are ca rezultat 2. Să mai observăm că în ce privește înmulțirea și împărțirea, dacă lucrăm cu elemente de tipuri diferite, atunci, prin schimbarea ordinei unor factori, putem obține rezultate diferite. De exemplu, valorile expresiilor $L * M * X$, $(I/J) * C$ pot fi diferite de $L * X * M$, respectiv de $I * C/J$.

Referitor la operația de ridicare la putere trebuie precizat faptul că baza poate fi întreagă, reală sau complexă, iar în ceea ce privește exponentul există limitări: dacă baza este un număr întreg și exponentul întreg, atunci rezultatul este întreg; dacă ori baza ori exponentul este real, atuci rezultatul este real; dacă baza este un număr complex, atunci exponentul nu poate fi decît întreg și rezultatul este de tip complex.

Dacă avem o operație de ridicare la o putere întreagă, pozitivă, este bine să o scriem sub formă de înmulțire repetată, deoarece astfel necesită un timp de calcul mai scurt. De exemplu, expresia $A * A$ se calculează mai rapid decît $A ** 2$.

3.4.2. Expresii logice

Cea mai simplă expresie logică este determinată de un singur element logic și anume: o constantă logică, o variabilă logică, o variabilă logică cu indici, o funcție logică scrisă în paranteză, care are întotdeauna valoarea $\cdot TRUE \cdot$ sau $\cdot FALSE \cdot$. Expresii logice mai complicate se pot obține folosind *relații aritmetrice* și *operații logice*.

---

* Tipul A este mai „tare'' decît B dacă $A \supset B$.

Scanned with OKEN Scanner

Relațiile aritmetice se construiesc cu următorii operatori:

$>$ notată *.GT.* („Greater Than");
$\geqslant$ notată *.GE.* („Greater than or Equal to");
$<$ notată *.LT.* („Less Than");
$\leqslant$ notată *.LE.* („Less than or Equal to");
$=$ notată *.EQ.* („EQual to");
$\neq$ notată *.NE.* („Not Equal to").

O relație exprimă o condiție aritmetică ce poate fi falsă sau adevărată. Numai expresiile aritmetice de tip întreg sau real pot fi combinate cu operatorii menționați anterior.

Exemplu: dacă se dau următoarele declarări de tipuri:

```
INTEGER  A,B,C,
REAL  I,J,K,LA
LOGICAL  L,H
COMPLEX  C1, C2
```

atunci următoarele construcții sînt expresii logice valabile:

```
A.GT.2
(A + B — C).GE.1
(I ** J + K — LA) .EQ.Ø.
```

iar următoarele construcții sînt greșite:

L.LE.H (*L* și *H* sînt variabile logice),
I.LT.(*C*1—*C*2). (*C*1, *C*2 sînt de tip complex).

Fiind date două elemente *A* și *B*, care sînt constante sau variabile logice sau expresii logice construite ca mai înainte, se pot forma cu ele următoarele *operații logice*:

.NOT.A care este definit de tabela:

A	*.TRUE.*	*.FALSE*
.NOT. A	*.FALSE.*	*.TRUE.*

A.AND.B, care este definit de tabela

B \ *A*	*.TRUE.*	*.FALSE.*
.TRUE.	*.TRUE.*	*.FALSE.*
.FALSE.	*.FALSE.*	*.FALSE.*

A.OR.B, care este definit de tabela

B \ *A*	*.TRUE.*	*.FALSE.*
.TRUE.	*.TRUE.*	*.TRUE.*
.FALSE.	*.TRUE.*	*.FALSE.*

Scanned with OKEN Scanner

Două operații logice pot apare una după alta numai dacă a doua este *.NOT.* Sensul acestor operații este: *NOT* = nu, *AND* = și, *OR* = sau.

Dacă ne referim la declarările din exemplul precedent, atunci următoarele construcții sînt valabile:

$$(A.LT.B).\ AND.\ H$$
$$.NOT.\ ((J - K + LA).GT.I)$$
$$(C.EQ.5).\ OR.L$$

iar următoarele construcții sînt greșite:

A.AND.L (*A* nu este expresie logică),

.OR.H (*.OR.* nu are sens singur),

L.OR.AND.H (*OR.* și *AND.* sînt consecutive).

Ordinea de evaluare a expresiilor logice este următoarea: dacă în ele intervin și expresii aritmetice, atunci se evaluează întîi aceste expresii conform ordinei lor proprii; apoi se verifică relațiile *.LT.*, *.LE.*, *.EQ.*, *.NE.*, *.GT.*, *.GE.*; în sfîrșit, se efectuează operațiile logice în ordinea *.NOT.*, *.AND.*, *.OR.*

Se poate constata că într-o expresie logică nu este nevoie să fie evaluate toate părțile ei; de exemplu, în expresia *A.OR.B*, dacă se cunoaște că valoarea lui *A* este *.TRUE.*, atunci pentru orice *B* valoarea logică rămîne *.TRUE.*

În expresiile logice se pot folosi și paranteze pentru a indica ordinea în care să se efectueze operațiile logice; cînd sînt paranteze interioare unele altora, cele mai interioare se evaluează primele. Parantezele pot schimba rezultatul expresiei; de exemplu, dacă *A* ia valoarea *·FALSE·* și *B* ia valoarea *·TRUE·*, atunci expresia

$$.NOT.\ (A.OR.B)$$

ia valoarea *.FALSE.*, în timp ce expresia

$$.NOT.A.OR.B$$

ia valarea *.TRUE.*

3.4.3. Instrucțiuni de atribuire

Valoarea unei expresii aritmetice poate fi atribuită unei variabile numerice, iar valoarea unei expresii logice poate fi atribuită unei variabile logice. Instrucțiunea de atribuire are forma

nume = expresie.

Scanned with OKEN Scanner

Exemple: instrucțiunile

$Y = (((A(4) * X + A(3)) * X + A(2)) * X + A(1)$

$Y = A(4) * X * * 3 + A(3) * X * * 2 + A(2) * X + A(1)$

$P(1) = (Q(1) * * 2 + R(1) * * 2)/4.Ø$

$P(1) = Ø.25 * (Q(1) * Q(1) + R(1) * R(1))$

sînt instrucțiuni de atribuire aritmetică, primele două, și respectiv ultimele două, sînt echivalente între ele.

O primă remarcă trebuie făcută în legătură cu instrucțiunea de atribuire; $A = B$ înseamnă că se calculează expresia aritmetică B și se pune rezultatul la locația lui A. În acest fel, în FORTRAN, are sens o relație de atribuire de forma

$$X = X + 2.$$

care are ca efect mărirea lui X cu 2.

Să vedem acum cum se face atribuirea aritmetică, în funcție de tipul variabilei din stînga și expresiei din dreapta instrucțiunii de atribuire. Dacă variabila și expresia au același tip, atunci nu este necesară nici o discuție.

Dacă variabila este de tip real și expresia de tip întreg, atunci se evaluează mai întîi expresia, iar valoarea întreagă obținută se transformă în valoare reală și după aceea se atribuie. Dacă variabila este de tip întreg și expresia de tip real, atunci se evaluează expresia, se ia partea întreagă a rezultatului și se atribuie variabilei.

Cînd variabila este complexă și expresia este reală, atunci se atribuie valoarea expresiei părții reale a variabilei, partea imaginară rămînînd zero. Dacă expresia este întreagă, atunci valoarea ei este mai întîi transformată în valoare reală și apoi atribuită conform regulei precedente. În sfîrșit, dacă variabila este reală și expresia este complexă atunci se calculează mai întîi expresia și apoi partea ei reală se atribuie variabilei. (Dacă variabila este întreagă, atunci are loc mai întîi convertirea precedentă complex-real, iar apoi în mod obișnuit convertirea real-întreg; deci variabilei i se atribuie partea dinainte de virgulă a părții reale a valorii expresiei complexe).

Exerciții

1. Să se construiască expresia aritmetică FORTRAN, corespunzătoare expresiei aritmetice

$$(5x^7 + (4x - 3)^2) : (x^2 - x + 1).$$

Scanned with OKEN Scanner

2. Se dă expresia

$$A + I/J + 2 * ((I + J) * A + 1) - A ** I$$

Să se calculeze valoarea ei dacă în prealabil s-au atribuit valorile $A = 1$, $I = 3$, $J = 2$.

Care este valoarea expresiei dacă J este de tip real?

3. Să se găsească greşelile din următoarea construcţie

$$A.B + - X * Y + 7{,}5 * (A ** X + Y$$

Să se corecteze greşelile astfel încît ea să devină construcţie valabilă.

4. Se dă următoarea expresie logică

$$.NOT.((X - 2).\ LT.\ (Y - 3.)).\ AND.(X + 1).\ GT.Y.OR.$$
$$.NOT.\ (X + Y).\ EQ.\ Ø.)$$

Să se afle valoarea ei dacă în prealabil s-au făcut atribuirile $X = 3$, $Y = 1$.

5. Să se indice ordinea operaţiilor din următoarea expresie

$$X.\ GT.Y\ .AND.\ NOT.\ HYP.\ OR.\ VAL$$

şi să se afle valoarea ei cînd $X, Y = 3.$, iar HYP şi VAL iau valoarea $.FALSE.$

3.5. FUNCŢII STANDARD

Există mai multe funcţii standard care pot fi incluse direct în expresii aritmetice (în englezeşte, ele se mai numesc *„BUILT — IN FUNCTIØNS"*). În privinţa acestor funcţii există oarecare diferenţe între diversele calculatoare, de aceea noi vom prezenta mai întîi numai pe acele care sînt aceeptate de cele mai multe compilatoare FORTRAN. (În ce priveşte *sintaxa* (scrierea) lor ne vom referi tot la IBM/360).

Cele mai obişnuite funcţii aritmetice sînt *rădăcina pătrată, cosinus, sinus, logaritmul natural* şi *funcţia exponenţială*. Ele se scriu de obicei astfel $SQRT(X)$, $COS(X)$, $SIN(X)$, $ALOG(X)$, $EXP(X)$, numele X notînd argumentul lor. Argumentul este întotdeauna real, iar valorile sînt de asemenea de tip real.

Scanned with OKEN Scanner

Alte funcţii se folosesc pentru convertirea unor valori sau expresii dintr-un tip în altul. De exemplu,

$$FLOAT(L)$$

transformă numărul întreg L într-un număr real, iar

$$INT(X)$$

transformă numărul real X în partea sa întreagă.

Dacă într-un calcul intervine valoarea absolută a unei variabile reale X, atunci ea se poate introduce în expresii aritmetice prin funcţia

$$ABS(X)$$

Depinzînd de calculator, se poate folosi şi funcţia

$$SIGN(X)$$

care defineşte semnul lui X; pentru $IBM/360$ se foloseşte funcţia $SIGN(X, Y)$, definită ca sgn (Y) înmulţit cu $|X|$.

Unele complilatoare FORTRAN au funcţii care definesc maximul a mai multor numere. De exemplu, funcţia

$$MAX(I,J,K)$$

defineşte maximul celor trei valori întregi; ea are tipul întreg. La fel, funcţia

$$MIN(I,J,K)$$

defineşte minimul. (Numărul argumentelor funcţiilor MIN şi MAX este $\geqslant 2$). O altă funcţie folosită uneori este

$$MOD(I,J)$$

care constă în a lua pe I modulo J; această funcţie poate fi definită de relaţia

$$MOD(I,J) = I - (I/J)*J.$$

În definiţiile de mai înainte tipul argumentului şi funcţiei sînt determinate de convenţia menţionată în § 3.3. Există însă, depinzînd de calculator, şi funcţii de alte tipuri. Ele se definesc la fel, numai, că în scriere se adaugă în faţa numelui

litera A pentru tipul real;
litera I pentru tipul întreg;
litera C pentru tipul complex;
litera D pentru tipul dublă precizie.

Scanned with OKEN Scanner

Exemple.

$DEXP(X)$ — înseamnă funcţia exponenţială evoluată cu dublă precizie;

$AMOD(X,Y)$ — înseamnă numărul $X - INT(X/Y) * Y$

$ISIGN(I)$ — înseamnă funcţia care defineşte semnul numărului întreg I;

$CLOG(X)$ — înseamnă logaritmul numărului complex X.

Desigur, construcţiile funcţiilor de diverse tipuri se pot face dacă ele au în primul rînd sens matematic. De exemplu, construcţia

$$CMAX(X1,X2)$$

nu are sens deoarece numerele complexe $X1$ şi $X2$ nu se pot compara.

În ce priveşte funcţiile standard, ele pot diferi mult, ca notaţie, tip sau varietate, de la un calculator la altul. Din această cauză, programatorul trebuie să folosească *neapărat* un manual FORTRAN al calculatorului particular pe care lucrează. Sistemele IBM/360 dispun de una din cele mai bogate liste de funcţii standard; dintre funcţiile obişnuite folosite de calculatoarele IBM menţionăm:

$ALOG10(X)$ — logaritmul zecimal;
$ARSIN(X)$ — arcsin x;
$ARCOS(X)$ — arccos x;
$ATAN(X)$ — arctg x;
$TAN(X)$ — tg x;
$COTAN(X)$ — ctg x;
$TANH(X)$ — tangenta hiperbolică;
$SINH(X)$ — sinus hiperbolic;
$COSH(X)$ — cosinus hiperbolic;

$ERF(X)$ — funcţia erorilor $= \frac{2}{\pi}\int_0^x e^{-u^2}\,du$;

$GAMMA(X)$ — funcţia gamma $= \int_0^\infty u^{x-1}e^{-u}\,du = \Gamma(x)$;

$ALGAMA(X)$ — $\ln\Gamma(x)$;

$CABS(Z)$ — $|Z|$, $Z = a + ib$;

$DIM(X,Y)$ — X — min (X,Y);

Scanned with OKEN Scanner

REAL(Z) — Re (Z), $Z = a + ib$;

AIMAG(Z) — Im (Z), $Z = a + ib$;

CONJG(Z) — $\bar{Z} = a - ib$, cînd $Z = a + ib$.

Tipul acestor funcţii este real cu excepţia ultimei care defineşte conjugatul unui număr complex. Multe din aceste funcţii au după caz, formă şi pentru dublă precizie.

Exerciţii

1. Să se programeze expresia

$$\sin^2(x^3 - x + 1) + e^{-3x}\cos(2x - 3) + \sqrt{e^{5x+1}} \cdot \ln(x^2 + 5).$$

2. Să se programeze expresia

$$2\cos x + 3\,\mathrm{tg}\,2x + [\mathrm{tg}\,(\ln(n^2 + 3))]\sin(\ln x),$$

unde prin $[V]$ se notează partea întreagă a lui V. (Se presupune că există funcţiile standard corespunzătoare).

3. Să se programeze expresia

$$x \equiv ay (\mathrm{mod}\ m),$$

presupunîndu-se mai întîi că x şi y sînt întregi, apoi că x şi y sînt reali (m şi a sînt întregi). (A se vedea funcţia *AMOD*).

3.6. TRANSFER CONDIŢIONAT ŞI NECONDIŢIONAT

Aşa cum am mai menţionat, în timpul execuţiei unui program, instrucţiunile sînt în general considerate în ordinea în care sînt scrise. Totuşi în multe sistuaţii este nevoie ca această ordine să fie alterată, fapt care se realizează prin *instrucţiuni de control.*

Instrucţiunile de control transferă controlul în mod condiţionat sau necondiţionat dintr-un loc al programului în altul. Legat de aceste instrucţiuni, se pune în evidenţă importanţa noţiunii de *număr al unei instrucţiuni* (vezi, de ex. § 3.2).

3.6.1. Transfer necondiţionat

Instrucţiunea de acest tip este de forma

GO TO n

Scanned with OKEN Scanner

unde n este un număr de instrucțiune; ea are ca rezultat continuarea programului cu instrucțiunea care are numărul de instrucțiune n. Transferul de control se face în acest caz *necondiționat*.

De exemplu, instrucțiunea

GO TO 32

transferă controlul la instrucțiunea care are scris în coloanele 4 și 5 numărul 32.

3.6.2. Transfer condiționat

Transferul determinat de o condiție aritmetică sau logică se numește transfer condiționat. Instrucțiunile de transfer condiționat sînt de două tipuri și anume *IF* și *GO TO*.

Instrucțiunea IF aritmetic este de forma

IF (expresie aritmetică) n_1, n_2, n_3

și ea are efectul că transferă controlul la instrucțiunea cu numărul n_1, n_2 sau n_3, după cum expresia aritmetică este, respectiv, negativă, zero sau pozitivă. Ea poate fi combinată și cu un *GO TO* necondiționat.

Instrucțiunile cu numerele n_i, $1 \leqslant i \leqslant 3$ trebuie efectiv să existe; în plus, instrucțiunea care urmează lui *IF* aritmetic *trebuie* să aibă un *număr*, căci altfel controlul nu se poate referi la ea și nu poate fi deci executată.

Atît în legătură cu instrucțiunile *IF* (= dacă) cît și cu cele de tip *GO TO* (= transfer la) se poate folosi și *instrucțiunea vidă*. Ea este de forma

n CONTINUE,

unde n este un număr de instrucțiune; ea nu exprimă nici o acțiune pe care s-o ia programul, ci este folosită numai ca o instrucțiune de *legătură* (*conector*) într-un program.

Exemplul 1. Pentru calculul funcției

$$F(x) = \begin{cases} \operatorname{tg} x, & \text{dacă } x < \frac{2}{3}, \\ x\, e^x, & \text{dacă } x = \frac{2}{3}, \\ \ln x, & \text{dacă } x > \frac{2}{3}, \end{cases}$$

Scanned with OKEN Scanner

se poate folosi următorul segment de program

```
    IF(X — 2./3.)4,5,6
4   F = SIN(X)/COS(X)
    GO  TO  7
5   F = X * EXP(X)
    GO  TO  7
6   F = LOG(X)
7   CONTINUE
```

Dacă, de exemplu, în program era nevoie să calculăm $y = 2F(x) + 3$, atunci, în loc de instrucţiunea vidă, numerotată cu 7, se putea folosi instrucţiunea

```
7   Y = 2. * F + 3.
```

Instrucţiunea IF logic are forma generală:
IF (expresie logică) instrucţiune
şi efectul ei este următorul: dacă expresia logică este adevărată (*.TRUE.*), atunci este executată instrucţiunea care urmează (în aceeaşi linie), iar dacă expresia este falsă, atunci se execută instrucţiunea din linia următoare a programului (se sare peste cea din linie cu *IF*). Instrucţiunea din aceeaşi linie cu *IF* trebuie să fie diferită de un alt *IF* logic sau de instrucţiunea *DO* (vezi § 3.9).

Exemplul 2. Să se scrie instrucţiunile unui program care calculează expresia $Z = \sqrt{3F^2(x) + 4}$, cu $F(x)$ definit de relaţia

$$F(x) = \begin{cases} x^2 e^{-x}, & \text{dacă } x \leqslant 0, \\ x + \ln T(x), & \text{dacă } x > 0, \text{ şi } x \in D_+, \\ x\, e^{-T(x)}, & \text{dacă } x > 0 \text{ şi } x \in D_-, \end{cases}$$

unde $D_+ = \{x/T(x) > 0\}$, $D_- = \{x/T(x) \leqslant 0\}$, iar $T(x) = x^2 - bx - c$, $(b,c > 0)$.

Soluţie. Trinomul $T(x)$ are două rădăcini reale, pe care le notăm cu $X1$ şi $X2$; segmentul de program este

```
    T = X * X — 9. * X + 2Ø
    IF(X.LE.Ø)  GO  TO  1Ø
    IF((X1.LE.X).AND.(X.LE.X2))  GO  TO  7
```


Scanned with OKEN Scanner

```
      F = X + LOG (T)
      GO   TO 11
    7 F = X * EXP(-T)
      GO   TO 11
   10 F = X * X * EXP(- X)
   11 Z = SQRT(3. * F * F + 4.).
```

Există și alte posibilități de a scrie instrucțiunile corespunzătoare problemei precedente. Vom comenta însă instrucțiunile scrise mai sus. Prima linie calculează valoarea trinomului T. Cea de-a doua linie testează relația $x \leqslant 0$ și dacă ea este adevărată, atunci se execută instrucțiunea 10; dacă $x \leqslant 0$ nu este adevărată, atunci se execută linia următoare care testează dacă $x_1 \leqslant x \leqslant x_2$ (adică $T \leqslant 0$); dacă $T \leqslant 0$, se trece la instrucțiunea 7 după a cărei execuție se trece la instrucțiunea 11; dacă $T > 0$, atunci se execută $F = X + LOG(T)$ (cea care urmează lui *IF* logic), iar instrucțiunea următoare *GO TO* 11 transferă de asemenea controlul la 11.

Instrucțiunea GO TO calculat („Computed GO TO") este de forma

$$GO \quad TO \quad (n_1, \ldots, n_k), i$$

unde n_j, $1 \leqslant j \leqslant k$, sînt numere de instrucțiuni, iar i este o variabilă întreagă fără indici. Această instrucțiune are efectul că dacă i ia o valoare $1 \leqslant i \leqslant k$, atunci controlul este transferat la instrucțiunea n_i; dacă $i > k$ sau $i < 1$, atunci nu se realizează nici un transfer de control, ci se execută instrucțiunea care urmează.

Exemplul 3. Se dă funcția

$$f(x) = \begin{cases} e^{-\frac{x^8}{2}} & 1 \leqslant x < 2, \\ e^{-x}, & 2 \leqslant x < 3, \\ xe^x, & 3 \leqslant x < 4, \\ 2\ \ln(x-3), & 4 \leqslant x \leqslant 5, \\ x, & x > 5 \end{cases}$$

și se dau numerele întregi ν_i, $1 \leqslant i \leqslant 5$.

Punînd $x_i = 0 \cdot 5 + i$, $S = \sum_i f(x_i) \cdot \nu_i$, să se scrie instrucțiunile FORTRAN pentru calculul lui S.

Scanned with OKEN Scanner

Soluție. Vom introduce listele $F(5)$ și $N(5)$, unde elementele lui N sînt ν_i cunoscute, iar elementele lui F trebuie calculate conform definiției lui $f(x)$. Instrucțiunile cerute sînt

```
    I = Ø
    X = Ø · 5
1   I = I + 1
    X = X + 1
    GO TO (3, 5, 7, 8, 9),I
    GO TO 1Ø
3   F(I) = EXP(−X * X/2.)
    GO TO 1
5   F(I) = EXP(−X)
    GO TO 1
7   F(I) = X * EXP(X)
    GO TO 1
8   F(I) = 2. * ALOG(X − 3.)
    GO TO 1
9   F(I) = X
    GO TO 1
```

$$1Ø\ \ S = F(1)*N(1) + F(2)*N(2) + F(3)*N(3) + F(4)*N(4) + F(5)*N(5)$$

Se puteau folosi și alte instrucțiuni pentru calculul sumei S, însă ele ar fi complicat scrierea. Instrucțiunea 1Ø poate fi scrisă mai simplu cu ajutorul unei instrucțiuni *DO* (vezi § 3.7).

Instrucțiunea GO TO atribuit („Assigned *GO TO*"). Ea este de forma

ASSIGN i *TO* m

.
.
.

GO TO m, $(n_1, \ldots, n_k)$

unde i este un număr de instrucțiune executabilă, m este o variabilă întreagă fără indici și $n_1, \ldots, n_k$ sînt numere de instrucțiuni executabile; i trebuie să ia una dintre valorile $n_1, \ldots, n_k$. Instrucțiunea *ASSIGN nu poate* fi înlocuită cu o instrucțiune de forma

$$M = I.$$

Scanned with OKEN Scanner

Exemplul 4. Să se scrie, folosind instrucțiunea *GO TO* atribuit, segmentul de program pentru calculul lui S de la exemplul 3.

Soluție. Folosind notațiile de la exemplul 3, segmentul de program căutat este

```
   X = Ø · 5
   ASSIGN 3 TO I
1  X = X + 1.
   GO TO I, (3, 5, 7, 8, 9)
   GO TO 1Ø
3  F(1) = EXP(— X * X/2.)
   ASSIGN 5 TO I
   GO TO 1
5  F(2) = EXP(—X)
   ASSIGN 7 TO I
   GO TO 1
7  F(3) = X * EXP(X)
   ASSIGN 8 TO I
   GO TO 1
8  F(4) = 2. * LOG(X — 3.)
   ASSIGN 9 TO I
   GO TO 1
9  F(5) = X
1Ø (același ca la exemplul 3).
```

Instrucțiunile *GO TO* pot fi înlocuite uneori cu instrucțiuni *IF*.

Exerciții

1. Să se scrie instrucțiunile FORTRAN pentru calculul funcției

$$f(x) = \begin{cases} x^2 + x + 1, & \text{dacă } x < 5, \\ 7x - 4, & \text{dacă } x = 5, \\ 2^x - 1, & \text{dacă } x > 5. \end{cases}$$

Scanned with OKEN Scanner

2. Să se scrie instrucţiunile pentru calculul valorilor funcţiei

$$f(x) = \operatorname{sgn}\ x\ g(x),\ g(x) = \begin{cases} 2x \cos x, & \text{dacă } |x| < \frac{\pi}{2}, \\ \operatorname{tg} \frac{x}{2} + e^x, & \text{dacă } \frac{\pi}{2} \leqslant |x| < \pi, \\ xe^{-x}, & \text{dacă } |x| > \pi. \end{cases}$$

3. Să se precizeze care din următoarele instrucţiuni sînt executabile:

$X = -2.$
$I = \emptyset$

1 $X = X + 3$
$I = I + 2$
GO TO (4, 5, 6), I
$F = 2*X + 1.$
GO TO 7

4 $G = X*X + X + 1.$
GO TO 1

5 $G = 3.*\ X*X - 1.$
GO TO 1

6 $G = 2\cdot*X$
GO TO 1

7 $Y = 2*F + 3*G.$

Care este valoarea lui Y?

3.7. OPERATORUL DE CICLARE

Una din situaţiile frecvent întîlnite în executarea unui program este necesitatea de a repeta de un număr de ori un anumit şir de operaţii. Instrucţiunea care realizează controlul unei astfel de situaţii este *un ciclu*, *buclă* sau *spiră*.

Scanned with OKEN Scanner

Forma generală a unei astfel de instrucțiuni este

$$DO\ n\ I = m_1,\ m_2,\ m_3$$

unde n este un *număr de instrucțiune* (vezi § 2), I este o variabilă întreagă fără indici și m_1, m_2, m_3 sînt întregi *pozitivi* sau variabile întregi.

Instrucțiunea *DO* indică executarea, cel puțin o dată, a instrucțiunilor ce o urmează pînă la instrucțiunea cu numărul n inclusiv; acest ciclu de operații este executat pentru $I = m_1$, $I = m_1 + m_3$, $I = m_1 + 2m_3$, ..., atît timp cît $I \leqslant m_2$ (din această cauză m_1 se numește *valoare inițială*, m_2 valoare *test* și m_3 *creștere*). Mai precis, numărul de execuții ale unei bucle este

$$\left[\frac{m_2 - m_1}{m_3}\right] + 1.$$

Dacă $m_2 < m_1$, atunci executarea buclei are loc o singură dată. Cînd $m_3 = 1$, el poate fi omis în scriere.

Instrucțiunea *DO* este cel mai frecvent utilizată în legătură cu masivele.

Exemplu. Să presupunem că se dau două liste $X(1ØØ)$ și $Y(1ØØ)$, de mărime $1ØØ$ fiecare, și ne interesează să calculăm sumele ΣX_i^2, ΣY_i^2, $\Sigma X_i Y_i$. Segmentul de program care realizează acest lucru (presupunînd că listele sînt deja memorate) este următorul

```
    SUMX2 = Ø
    SUMXY = Ø
    SUMY2 = Ø
    DO 5 I = 1, 1ØØ
    SUMX2 = SUMX2 + X(I)*X(I)
    SUMY2 = SUMY2 + Y(I)*Y(I)
5   SUMXY = SUMXY + X(I)*Y(I)
```

Semnificația numerelor anterioare folosite este evidentă. Efectul pe care îl are instrucțiunea *DO* este că execută de 100 de ori instrucțiunile pînă la instrucțiunea cu numărul 5 inclusiv; calculul pornește cu I egal cu 1, iar după 100 de execuții ale buclei I este egal cu 100.

Buclele sau ciclurile pot fi incluse una în alta. De exemplu, dacă avem două masive bidimensionale $A(5,\ 1Ø)$ și $B(5,\ 1Ø)$, atunci putem calcula sumele pătratelor elementelor lor și suma produselor elementelor lor conform instrucțiunilor

Scanned with OKEN Scanner

```
      SUMAB = Ø
      SUMA2 = Ø
      SUMB2 = Ø
      DO 5 I = 1,5
      DO 5 J = 1,1Ø
      SUMA2 = SUMA2 + A(I, J)*A(I, J)
      SUMB2 = SUMB2 + B(I, J)*B(I, J)
5     SUMAB = SUMAB + A(I, J)*B(I, J)
```

Despre buclele incluse una în alta se mai spune că sînt *cuibărite* una în alta. Mai multe bucle se pot determina sau nu cu aceeaşi instrucţiune.

În legătură cu folosirea buclelor sînt necesare următoarele precizări:

1. Nu este permis ca parametrii (I, m_1, m_2, m_3) ai buclei să fie schimbaţi în instrucţiuni *interioare* spirei; ei se pot schimba numai în afara spirei.

2. Oricînd este permis un transfer din interior în afara buclei; transferul din interiorul în interiorul buclei este de asemenea permis, dar *nu* este permis transferul din afara buclei spre interior.

3. Dacă în interiorul unei bucle începe o altă buclă (cu o linie *DO*), atunci această ultimă buclă nu trebuie să se termine mai departe de prima; deci cea de-a doua spiră trebuie să fie conţinută în prima. În acest caz, cele două bucle se pot termina cu aceeaşi instrucţiune (vezi exemplul precedent referitor la spire cuibărite).

4. Dacă mai multe bucle (cuibărite) se termină cu aceeaşi instrucţiune, atunci instrucţiunea respectivă este considerată ca aparţinînd *celei mai interioare* dintre spire; cu alte cuvinte, dacă vrem să transferăm controlul la această instrucţiune, trebuie să facem acest transfer numai plecînd din interiorul celei mai *adînci* spire (celei mai interioare).

5. Ultima instrucţiune a unei bucle trebuie să fie o instrucţiune executabilă; ea nu poate fi de tipurile *GO TO*, sau de tipurile *IF*, nu poate fi nici un alt *DO* şi de asemenea nu poate fi: *PAUSE*, *STOP*, *RETURN* (vezi secţiunile următoare). Pentru a evita terminarea unei bucle în instrucţiuni nepermise de tipul celor menţionate anterior, se poate folosi instrucţiunea *CONTINUE*. (Aici se vede de fapt necesitatea instrucţiunii *CONTINUE* introdusă în § 3.8).

Exemple de construcţii corecte sînt următoarele:

Scanned with OKEN Scanner

Exemplul 1.

```
   DO 1 I = 1, 1Ø
7  IF(A(I) — B(I)) 6, 1, 1
6  A(I) = A(I) + 1·Ø
   B(I) = B(I) — 2·Ø
   GO TO 7
1  CONTINUE
```

Explicație: Aici instrucțiunea vidă a fost folosită pentru a evita terminația în *GO TO*.

Exemplul 2.

```
   DO 1 I = 1, 1Ø
   IF(A(I) — B(I))2, 3, 3
2  A(I) = B(I)
   GO  TO 1
3  A(I) = C(I)
1  CONTINUE
```

Explicație: Aici instrucțiunea vidă *CONTINUE* ne permite să ocolim instrucțiunea 3.

Exerciții

1. Ce greșeli sînt în următoarea construcție?

```
   F = Ø.
   DO 1Ø·I = 1,2Ø
   IF(X — L.) 7, 8, 8
7  F = F + X * SIN(X)
   DO 8 J = 1,  5
   F = F + X * X
   I = J + L
8  A(I) = X + F
```

2. Ce greșeli sînt în următoarea construcție?

```
   DO 1Ø I = 1,  5
   B(I) = A(I) + C(I)
   IF(B(I) — A(I)) 8, 9, 9
```


4—23

Scanned with OKEN Scanner

```
8 DO 9 J = 1, 5
  D(I) = B(I)*B(I)
1Ø CONTINUE
9 A(I) = D(I) + C(I) - B(I)
```

3. Să se scrie instrucțiunile care transformă o matrice pătratică 30×30 în transpusa ei.

4. Presupunînd că numerele $x_1, \ldots, x_n$ sînt introduse în calculator (eventual sînt deja memorate), să se scrie instrucțiunile pentru calculul mediei aritmetice, dispersiei și deviației standard; se folosesc formulele cunoscute

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} x_i, \qquad s^2 = \frac{1}{n} \sum_{i=1}^{n} x_i^2 - \left(\frac{1}{n} \sum_{i=1}^{n} x_i \right)^2.$$

Ce declarare trebuie făcută pentru a se evita eroarea $s^2 < 0$?

3.8. INSTRUCȚIUNI DE INTRARE ȘI IEȘIRE

Această secțiune se ocupă de cele mai simple forme ale instrucțiunilor FORTRAN pentru introducerea datelor și scoaterea rezultatelor din calculator, fără a se menționa toate facilitățile limbajului în această direcție. În fapt, vor fi considerate numai instrucțiunile legate de *citirea* datelor de pe cartele și scrierea rezultatelor pe hîrtia de imprimantă.

Trebuie menționat de la început că instrucțiunile de *intrare* și *ieșire* sînt puternic dependente de configurația de calcul utilizată. Această secțiune are în vedere numai *instrucțiunile* secvențiale de intrare și ieșire cu *FORMAT* pentru sistemele IBM 360. Instrucțiunile secvențiale de intrare sau ieșire au ca efect transferul unor date așezate succesiv în *unități de înregistrare*. În cazul citirii de pe cartele, unitatea de înregistrare este mulțimea de date de pe o cartelă, iar în cazul tipăririi pe hîrtia de imprimantă, unitatea de înregistrare este mulțimea de date de pe o linie orizontală. Unitatea de înregistrare va fi numită pe scurt înregistrare.

3.8.1. Instrucțiunea de intrare *READ* cu *FORMAT*

Această instrucțiune este folosită pentru a prelua date de pe cartele, spre a le introduce în memoria calculatorului. Forma sa este

READ (a, b) listă

Scanned with OKEN Scanner

unde a este un întreg pozitiv care reprezintă un *cod de referință* al unei mulțimi de date, iar b este *numărul* unei instrucțiuni *FORMAT* care descrie înregistrările ce trebuie citite. Lista conține numele variabilelor sau masivelor care trebuie citite, despărțite prin virgulă; instrucțiunea *READ* va *citi* de pe cartelă datele și le va depune în memorie la adresele variabilelor din listă. Codul (numărul) de referință precizează care periferic din configurația de calcul va realiza citirea la sistemele IBM 360. Citirea de pe cartele se realizează cu codul $a = 1$.

Exemplul 1. Instrucțiunea

$$READ\ (1,12)\ X, Y, ((W(I, J), I = 1, 5), J = 1, 1Ø)$$

are ca efect citirea de pe cartele a valorilor variabilelor X și Y și a matricei W de 5×10 elemente (a se remarca scrierea în listă a masivului W). Descrierea datelor care se citesc (și care au tip real) este dată de instrucțiunea *FORMAT* cu numărul 12. Elementul $((W(I, J), I = 1, 5), J = 1, 1Ø)$, care apare în listă, definește un *ciclu* al operației de citire (cunoscut sub numele de „*DO implicit*"). El asigură citirea coloanelor succesive ale masivului, care vor fi memorate în ordinea citirii, adică $W(1, 1), W(2, 1), \ldots, W(5, 1), W(1, 2)$; citirea se poate face și cu instrucțiuni de forma

$$DO\ 1\ J = 1, 1Ø$$
$$DO\ 1\ I = 1, 5$$
$$1 \cdot READ(1, 2)W(I, J)$$

unde 2 este numărul instrucțiunii *FORMAT*, dar execuția acestor instrucțiuni consumă un timp de calcul mai mare. Trebuie remarcat faptul că datele citite se pot memora la locații de memorie nu neapărat vecine.

3.8.2. Instrucțiunea *WRITE* cu *FORMAT*

Această instrucțiune este folosită pentru a transfera date din memorie pe un mediu exterior și are forma

$$WRITE\ (a, b)\ listă$$

Semnificația elementelor a, b și a listei este aceeași ca pentru instrucțiunea *READ*. În cazul scrierii pe hîrtiea de imprimantă, $a = 3$.

Exemplul 2. Instrucțiunea

$$WRITE\ (3, 5)\ U, W,(X(J), J = 1, 7)$$

Scanned with OKEN Scanner

are ca efect scrierea pe hîrtia de imprimantă a valorilor care se găsesc în memorie la locaţiile variabilelor U şi W şi a vectorului X de 7 componente. Descrierea modului în care se tipăresc aceste date este dată de instrucţiunea *FORMAT* cu numărul 5.

Instrucţiunile *READ* şi *WRITE* cu *FORMAT* se pot referi şi la înregistrări pe alte memorii externe, diferite de cartela perforată.

Există şi alte forme ale acestor instrucţiuni şi anume *READ* şi *WRITE* fără *FORMAT* sau *READ* şi *WRITE* cu *NAMELIST*, dar asupra lor nu insistăm aici (vezi şi Cap. IV).

3.8.3. Instrucţiunea *FORMAT*

Operaţiile de intrare sau ieşire se realizează numai dacă datele cu care acestea lucrează sînt precis descrise. Transmiterea datelor se poate face fie sub controlul unei instrucţiuni *FORMAT*, fie fără un astfel de control.

Instrucţiunea *FORMAT* (care poate apare oriunde în program) este utilizată în legătură cu instrucţiunea *READ* şi *WRITE* şi precizează *structura* înregistrării datelor care intervin în execuţia unui program *FORTRAN*.

În instrucţiunea *FORMAT* *cîmpurile* de date sînt descrise cu *coduri* format, despărţite prin virgule. Forma generală a instrucţiunii *FORMAT* este următoarea

$$XXXX\ FORMAT\ (c_1, \ldots, c_n)$$

unde *XXXX* este un număr de instrucţiune scris în coloanele 2—5 avînd cel puţin o cifră semnificativă, iar $c_1, \ldots, c_n$ sînt coduri. Codurile se folosesc pentru a face distincţie între tipurile de date care pot compune o înregistrare. Astfel, codurile *primare* precizează dacă este vorba de date numerice (reale, întregi), logice sau literale. Vom descrie mai întîi cîteva din *codurile primare*.

Codul *I*. Acest cod descrie cîmpuri de numere întregi. Descrierea *unui* număr întreg se face prin codul *Il* unde *l* reprezintă numărul de cifre ale numărului. *La intrare*, blancurile sînt luate în considerare ca zerori. *La ieşire*, dacă numărul din memorie are mai mult de *l* cifre zecimale, atunci aceasta constituie o eroare şi deci numărul nu se va tipări (se vor tipări steluţe *). Dacă în schimb el are mai puţin de *l* cifre, atunci tipărirea se face prin *ajustarea la dreapta* (complectîndu-se deci numărul cu blancuri în faţă). Numărul *l* de cifre, inclusiv semnul, trebuie să fie astfel precizat încît să nu corespundă unui număr întreg ce nu se poate reprezenta în memorie.

Scanned with OKEN Scanner

Codul *F*. Acest cod descrie cîmpuri de numere reale scrise cu punct zecimal. Descrierea unui număr real se face prin codul de forma *Fl·z*, unde *l* reprezintă numărul de caractere din scrierea numărului (inclusiv punctul zecimal dacă există) iar *z* ($z \leqslant l$) este numărul de zecimale.

Codul *E*. Acest cod descrie tot cîmpuri de numere reale, dar care se scriu cu ajutorul unui exponent. Descrierea unui număr real în acest mod se face prin codul *El·z*, unde *l* reprezintă numărul de caractere al mantisei (inclusiv punctul zecimal, dacă există), iar *z* numărul de zecimale al acesteia.

Codul *D*. Acest cod descrie cîmpuri de numere reale în dublă precizie și se aseamănă cu codul *E*.

Se impun cîteva precizări în legătură cu codurile *E*, *F*, *D*.

La intrare, numărul trebuie să fie scris (perforat) ca număr real și, în funcție de cod, poate să aibă opțional pe *E* sau *D*. Punctul zecimal poate fi omis. De exemplu, numărul 2531, citit cu formatul *F*4. 1, va ajunge în memorie sub forma 253. 1. Cele trei coduri sînt și interschimbabile în ce privește intrarea. De exemplu, citirea numărului 35. 1*E* + 24 se face cu oricare din codurile *E*, *F*, *D*.

La ieșire, datele cu codurile *E* sau *D* sînt tipărite astfel: un punct zecimal, cele *z* cifre zecimale și un exponent *E* sau *D* care necesită patru poziții. Deci *l* = 1 (pentru semn) +1 (punctul zecimal) + *z* (numărul de cifre) +4 (pozițiile pentru exponent). Codul *F* la ieșire trebuie să prevadă spațiu suficient pentru partea întreagă. Lungimea cîmpului *l* este în acest caz: *l* = partea întreagă +1 (punct zecimal) + *z* (număr de cifre). În general, dacă codurile *I*, *F*, *E*, *D* la ieșire nu au *l* suficient de mare pentru a reprezenta numărul, se tipărește de *l* ori caracteterul* (așa apare eroarea la IBM 360).

Codul *L*. Acest cod descrie cîmpuri de date logice. O valoare logică este descrisă de codul *Ll*, unde *l* este numărul de caractere cu care se exprimă acea constantă logică.

Dacă la *intrare* primul caracter din cîmpul descris cu codul *L* este *T* sau *F*, atunci în memorie se depun valorile *.TRUE.* sau *.FALSE.* pentru constanta logică. Cînd cîmpul celor *l* caractere constă din blancuri, se preia valoarea *.FALSE.* și se depune în memorie.

La ieșire lucrurile se petrec în mod asemănător. Dacă variabila logică, ce se scoate la imprimantă, are valoarea *.TRUE.* sau *.FALSE.*, se tipărește *T* sau *F* precedat de *l* — 1 blancuri.

Coduri pentru constante literale. Datele literale pot apare în instrucțiunea *FORMAT* în două moduri diferite: scrise între două apostrofuri, ori după un cod *H*. Codul *H* este de forma *nH*; citirea sau scrierea unei constante literale cu codul *H* se face luînd în considerare cele *n* caractere de după codul *H*. De exemplu, cu codul 3*H OM* se vor scrie numai caracterele *OM* precedate de un blanc. Același lucru se

Scanned with OKEN Scanner

realizează cu codul cu apostrofuri și anume *'OM'*. Primul caracter după *H* sau primul caracter de după primul apostrof au rolul de a controla *cariajul* la ieșire. De exemplu, blanc sau zero înseamnă scrierea pe o nouă linie a imprimantei, respectiv se lasă un rînd liber, iar 1 înseamnă scrierea pe prima linie a paginii următoare hîrtiei de imprimantă.

Dacă într-un astfel de cod se cere să se tipărească semnul ' (apostrof) atunci se vor folosi două (sau trei) apostrofuri consecutive.

Codul *X*. Acest cod se folosește pentru a lăsa spațiu liber. El are forma *nX*, unde *n* este numărul de spații libere. Dacă un cod *X* apare la o instrucțiune *de intrare*, atunci înseamnă că *n* poziții de pe cartelă nu vor fi citite, iar caracterele care urmează după aceste *n* poziții vor fi atribuite următoarei variabile, care se citește cu următorul cod *FORMAT*. În mod asemănător la ieșire, cînd apare codul *nX*, se vor lăsa *n* blancuri pînă la tipărirea variabilei descrise de următorul cod *FORMAT*.

Codul *A*. Codul *A* se folosește la transmiterea datelor numerice sau nenumerice, caracter cu caracter. Fiecare caracter are codul său specific de reprezentare în memoria internă, într-un bait. Datele numerice sînt convertite cifră cu cifră, fără să se acorde o semnificație numerică cîmpului rezultat, prin juxtapunerea mai multor baiți. Deci codul *A* nu se folosește cu operațiile aritmetice. Numărul maxim de caractere care poate fi memorat sau extras din memoria internă este determinat de lungimea listei de variabile din instrucțiunea de intrare-ieșire. (Ultima aplicație din capitolul V este un exemplu de folosire a codului *A*).

Codurile *primare I, F, E, D, L, A*, folosite mai sus, se referă la descrierea unui singur cîmp de date. Dacă însă este nevoie să se citească sau scrie *n* cîmpuri consecutive de date de același fel, atunci se va scrie în fața codului respectiv numărul întreg *n*. Se obține astfel un cod compus.

O instrucțiune *FORMAT* poate descrie o înregistrare dacă codurile primare sau compuse sînt despărțite prin virgule și numărul total de caractere care rezultă din aceste coduri nu depășește numărul maxim de caractere admis de înregistrare (80 caractere la intrare, adică numărul de coloane al cartelei și 130 caractere la ieșire, adică numărul maxim de caractere ce pot fi tipărite în linie de către imprimantă). Dar o instrucțiune format poate să descrie și mai multe înregistrări, separîndu-le prin caracterul /. Dacă un grup de coduri se cere repetat de *n* ori, atunci acesta se include în paranteze și se scrie *n* în fața parantezei.

În sfîrșit, instrucțiunea *FORMAT* care însoțește o instrucțiune de intrare sau ieșire trebuie să descrie forma fiecărei înregistrări de

Scanned with OKEN Scanner

date ce se va utiliza. Dacă sîntem în cazul mai multor înregistrări a căror formă se repetă, atunci este suficient dacă instrucțiunea *FORMAT* descrie o singură înregistrare.

Pentru a înțelege mai bine utilizarea principalelor coduri *FORMAT* descrise mai sus vom da cîteva exemple.

Exemplul 3. Dacă următoarele caractere sînt perforate pe trei cartele începînd din prima coloană

12.531.27351*ABC*3581734
Ø1.895.7ØØ14*VAS*7Ø51345
21.354.31ØØ1*FAR*Ø51Ø821

atunci instrucțiunea

READ(1,1) (*A*(*I*), *B*(*I*), *N*(*I*), *M*(*I*), *I* = 1,3)
1 *FORMAT*(2*D*4.1,2(*I*4,3*X*))

va realiza citirea următoarelor date care se depun în modul adecvat în memoria calculatorului

A(*I*)	*B*(*I*)	*N*(*I*)	*M*(*I*)
12.5	31.2	7351	3581
1.8	95.7	Ø914	7Ø51
21.3	54.3	1ØØ1	51Ø.

Dacă H este o variabilă logică și ne referim tot la cartelele menționate, atunci instrucțiunea

READ (1, 2) *A*, *B*, *X*, *Y*, *U*, *V*, *H*, *I*, *J*, *K*
2 *FORMAT* (*F*5, 2, *D*5.3,5*X*2(*F*3.2)/*E*4.1,1 *XD*5.3,2 *XL*3,1*X*I 2, I1, I2)

va introduce în memorie valorile

A = 12.53 *B* = 1.273 *X* = 35.8 *Y* = 17.3
U = 21.3 *V* = 4.31Ø *H* = .*FALSE*. *I* = 51 *J* = 8
K = 21

(A se remarca faptul că *D*, *E*, *F* se pot interschimba la intrare ; semnul 1*X* înseamnă evitarea citirii unui caracter ; după codul *X* nu este neapărat nevoie de virgulă ; virgula și / sînt separatori, așa încît după / nu este nevoie să se pună virgulă).

Exemplul 4. În memoria calculatorului sînt memorate următoarele

A(*I*) = 2.8 3.2 7.4 3.4
ABAR = 4.2
H = .*TRUE*. (*H* este variabilă logică)
N = 2Ø

Scanned with OKEN Scanner

Atunci instrucţiunea

```
    WRITE (3,3) N, (A(I), I = 1,4), ABAR, H
3 FORMAT (' ', I3, 4      F5.1,      5XF4.1/' H = ', L7)
```

va tipări următoarele

```
— 2Ø — — 2.8 — — 3.2 — — 7.4 — — 3.4 — — — — — — 4.2
— H = — — — — — — T
```

(Liniuţele desemnează blancuri; tipărirea — inclusiv blancurile — începe cu prima poziţie din linia imprimantei).

Să presupunem acum că $A(I)$ reprezintă patru probe ale unui experiment şi că sînt $N = 2Ø$ astfel de experimente. Fiecare experiment este înregistrat pe o cartelă cu codul $4F4.1$. Se cere calculată media experimentului; dacă media depăşeşte valoarea $4.Ø$, atunci $H = .TRUE.$ Să se scrie instrucţiunile pentru prelucrarea celor $2Ø$ experimente şi să se tipărească rezultatele într-un tabel de forma:

Nr. Crt.	Probe	Media	H

Instrucţiunile sînt

```
    WRITE (3,1)
1 FORMAT (' ', (' — ')/'NR.CRT', 5X'—P—R—O—B—E',
  — 5X'MEDIA', 2X'H', 1X/34('—')) (*)
    DO 2 I = 1,2Ø
    READ(1,3) (A(J), J = 1,4)
3 FORMAT(4F4.1)
    ABAR = Ø
    DØ 4 J = 1,4
4 ABAR = ABAR + A(J)
    ABAR = ABAR/4.
    IF(ABAR — 4.)5,5,6
5 H = .FALSE.
    GO TO 2
6 H = .TRUE.
```

* Acest rînd este scris cu semnul de continuare pe coloana a 6-a.

Scanned with OKEN Scanner

2 *WRITE* (3,7) *I*, (*A*(*J*), *J* = 1,4), *ABAR*,*H*
7 *FORMAT*(' ' *I*7, 2*X*, 4*F*4. 1, *F*6.1,*L*4)

Instrucţiunea *FORMAT* cu numărul 1 tipăreşte capul de tabel specificat mai sus, iar cea cu numărul 7 defineşte modul de tipărire al rezultatelor.

Cunoştinţiele despre instrucţiunile de intrare şi ieşire prezentate aici sînt în general suficiente pentru a permite cititorului lucrul cu fişiere de date pe cartele. Trebuie subliniat din nou faptul că, pentru utilizarea corectă a acestor instrucţiuni, cititorul trebuie neapărat să consulte şi manualul de programare în FORTRAN al sistemului de calcul pe care-l utilizează.

Exerciţii

1. Pe două cartele sînt înregistrate din coloana 1 datele
123Ø.4531.25
47.1234.56.

a) ce valori vor avea în memorie variabilele citite cu instrucţiunea
3 *FORMAT*(I2, *F*4.1, *E*6.2/*F*4.1,*E*6.2)
READ (1,3)*N*, *A*, *B*, *C*, *D*

b) Dacă fişierul de date constă din cele două cartele, instrucţiunile
4 *FORMAT* (*I*3, 2*F*5.3//*F*4.1)
READ(1,4)*N*, *A*, *B*, *C*
sau
5 *FORMAT*(*I*3, 2*F*5.3/)
READ(1,5)*M*, *X*, *XX*
sînt corecte?

2. În memorie sînt numerele $N = 2Ø1$, $A = 45.23$, $B = 51.31$, $C = 23.2$. Ce va tipări instrucţiunea *WRITE* (3,4) *N*, *A*, *B*, *C* cu instrucţiunea 4 de la exerciţiul precedent.

3. Sînt corecte următoarele instrucţiuni *FORMAT*?
3 *FORMAT* ('*APA*/'/3*H VOL*//*F*6.1)
5 *FORMAT* (3(2*XF*5, 1), 5*X*4(2*XF*3.1)/
6 *FORMAT*(1Ø(1*XF*9.1), *I*8, 4*F*4.1)

4. Ce tipăresc instrucţiunile:
a) *WRITE* (3,4) *A*,*B*
4 *FORMAT*('*MEDIA* = ', *F*9.1/'*DISPERSIA* = ',*F*6.1)
dacă în memorie $A = 32.5$, $B = 7.6$?
b) *WRITE* (3,1) *DELTA*, *H*
1 *FORMAT*('*DELTA ESTE*', *F*6.2//' ', 2*X*'*H* = ', *L*3)
dacă în memorie *DELTA* = 75312.473, iar *H* = .*FALSE*.?

Scanned with OKEN Scanner

3.9. SUBPROGRAME. FUNCȚII ȘI SUBRUTINE

Instrucțiunile prezentate pînă acum permit construirea unui *program sursă*. Un program *FORTRAN* trebuie să conțină la început declarările tipurilor și listelor, iar apoi instrucțiunile care descriu soluția problemei. În plus, un program trebuie să mai conțină neapărat următoarele instrucțiuni:

a) *STOP*, care cînd apare în program precizează faptul că execuția programului se termină acolo. Din cauza instrucțiunilor *IF* sau *GO TO*, ea poate apare oriunde în lista de instrucțiuni a programului;

b) *END*, care se pune întotdeauna la sfîrșitul listei de instrucțiuni a programului sursă precizînd faptul că acel program se termină acolo. *END* este o instrucțiune ne-executabilă, care dă indicații compilatorului cînd se termină compilarea. Programul mai poate conține după cum am văzut și comentarii care să precizeze conținutul lui. Un program FORTRAN nu are în mod obligatoriu un nume.

În diversele programe FORTRAN se întîlnesc situații cînd anumite instrucțiuni ar trebui repetate în anumite puncte ale programului. Pentru a ușura scrierea unor astfel de programe, precum și pentru a micșora numărul lor de instrucțiuni se obișnuiește ca aceste programe să fie divizate în mai multe subprograme (rutine). Această divizare face ca dacă într-un anumit loc din program este nevoie de executarea instrucțiunilor unei anumite rutine, atunci printr-o instrucțiune ad-hoc se *cheamă* acea rutină (care de obicei poartă un nume) și nu se trece la executarea următoarei instrucțiuni din program pînă cînd nu se execută toate instrucțiunile rutinei. Există trei tipuri de astfel de subprograme și anume *funcția instrucțiune* („Statement Function"), *funcția subprogram* („*FUNCTION*") și *subrutina* („*SUBROUTINE*").

Pentru orice subprogram trebuie să avem în vedere două aspecte importante și anume *definirea* și *folosirea* ('chemarea' în programul principal).

Vom prezenta pe rînd cele trei tipuri de subprograme, avînd în vedere atît definirea cît și folosirea lor.

3.9.1. **Funcția instrucțiune** are forma generală

Nume (*listă de variabile*) = *expresie*.

'Numele' se construiește conform regulilor FORTRAN și el desemnează numele funcției; în paranteză trebuie să existe o listă de *variabile* care sînt argumentele funcției (ele se mai numesc și *argumente formale*); expresia din partea dreaptă a egalității definește operațiile ce trebuie

Scanned with OKEN Scanner

efectuate în program cînd apare numele funcţiei; valoarea expresiei este valoarea funcţiei.

Variabilele din listă nu pot fi variabilele cu indici. Expresia trebuie să fie de acelaşi tip cu cel al numelui funcţiei; ea nu trebuie să conţină nici o referire la funcţia însăşi.

Funcţia definită ca mai sus poate fi folosită în alte instrucţiuni ce o urmează, cu condiţia ca argumentele ei *curente* (actuale) să corespundă ca număr, ordine şi tip cu argumentele formale ale funcţiei; drept argumente actuale ale funcţiei pot fi orice expresii (aritmetice sau logice) de acelaşi tip cu argumentele formale.

Exemplul 1. Dacă programatorul trebuie să folosească de mai multe ori într-un program un trinom de gradul doi $T(x) = ax^2 + bx + c$, unde coeficienţii a, b, c pot varia de la o etapă la alta a calculelor, atunci el poate defini o funcţie de forma

$$TRINOM\ (A,\ B,\ C,\ X) = A*X*X + B*X + C$$

Dacă ulterior în program are nevoie să atribuie valoarea trinomului ai cărui coeficienţi sînt respectiv $F*F$, $G + 2.$, $N*H + 3.$, iar $X = 2$ unei variabile, atunci el poate folosi instrucţiunea de atribuire

$$Z = TRINOM\ (F*F,\ G + 2.,\ N*H + 3.,2.)$$

Funcţiunile instrucţiune sînt asemănătoare funcţiilor standard (vezi §3.5); deosebirea constă în faptul că funcţiile standard pot fi folosite în orice program, pe cînd funcţiile instrucţiune pot fi folosite numai în programele în care au fost definite. Ele se numesc funcţii instrucţiune deoarece sînt definite de o singură instrucţiune.

3.9.2. **Funcţia subprogram** este un subprogram FORTRAN, format din instrucţiunea *FUNCTION* urmată de alte instrucţiuni FORTRAN, printre care se află cel puţin o instrucţiune *RETURN*. Funcţia subprogram este un program independent de programul principal şi el este executat ori de cîte ori *numele* funcţiei apare în programul principal (vezi mai jos).

Instrucţiunea *FUNCTION* are următoarea formă

Tip FUNCTION nume (listă de argumente)

şi ea se pune la începutul subprogramului. Tipul poate fi întreg, real, complex sau logic şi el trebuie să fie declarat la început (precizarea cuvîntului 'Tip'), fie subînţeles conform convenţiilor FORTRAN. Numele funcţiei urmează după *FUNCTION*, iar lista din paranteze conţine argumentele formale ale funcţiei; argumentele pot fi variabile fără

Scanned with OKEN Scanner

indici, nume de liste (masive), nume de subrutine sau alte funcţii subprogram.

Deoarece *FUNCTION* este un subprogram independent, numerele sale de instrucţiune nu au nimic de-a face cu numerele de instrucţiune din programul principal.

FUNCTION poate conţine orice instrucţiune în afară de *SUBROUTINE* şi *FUNCTION*. Numele funcţiei *trebuie* să apară neapărat măcar într-o relaţie de atribuire (în partea stîngă) în subprogram.

Variabilele din instrucţiunea *FUNCTION* se numesc *parametri formali*. Cînd funcţia este folosită într-un program ('este chemată'), variabilele ei se numesc *argumente curente* sau *parametri actuali*.

Chemarea subprogramului *FUNCTION* de către programul principal este realizată printr-o instrucţiune de atribuire de forma

$$Z = nume \text{ (lista de parametri actuali)}$$

unde în loc de „nume" figurează numele funcţiei.

Diferenţa dintre funcţia instrucţiune şi funcţia subprogram constă atît în forma lor sintactică, cît şi în faptul că prima este definită de o singură instrucţiune. Între argumentele formale şi cele curente trebuie să existe corespondenţă în ce priveşte tipul, ordinea, numărul, etc.

Instrucţiunile *RETURN* şi *END* au rol similar instrucţiunilor *STOP* şi *END* din cadrul programului principal.

Instrucţiunea *RETURN* „întoarce" valoarea calculată a funcţiei programului care a chemat acea funcţie (am văzut mai sus că numelui funcţiei i se atribuie o valoare în timpul executării subprogramului).

Instrucţiunea *END* are exact aceeaşi semnificaţie ca într-un program principal (adică înseamnă sfîrşitul subprogramului, deci sfîrşitul compilării).

Exemplul 2. Fiind date două numere întregi r_0 şi r_1, $r_0 > r_1$, să se scrie o funcţie subprogram pentru determinarea celui mai mare divizor comun al lor.

Vom nota cele două numere *IRØ* şi *IR*1, iar funcţia căutată *ICMMDC* (*IRØ*, *IR*1). Vom aplica algoritmul lui Euclid ; notăm cu *IR*2 resturile succesive. Funcţia căutată are forma

```
                FUNCTION ICMMDC (IRØ, IR1)
              1 IR2 = IRØ - (IRØ/IR1)*IR1
C - - - - - - INSTRUCTIUNEA PRECEDENTA DETER-
                MINA
C - - - - - - RESTUL IMPARTIRII LUI IRØ PRIN IR1.
                IF(IR2)3, 4, 3
```


Scanned with OKEN Scanner

```
3 IRØ = IR1
  IR1 = IR2
  GO TO 1
4 ICMMCD = IR1
  RETURN
  END
```

Dacă în programul principal ne interesează să calculăm cmmdc a două numere M şi N $(M > N)$ şi să memorăm rezultatul la L, atunci putem realiza acest lucru chemînd subprogramul astfel

$$L = ICMMDC\ (M,\ N).$$

Condiţia ca $r_0 > r_1$ poate fi evitată dacă, înainte de instrucţiunea cu numărul 1, introducem instrucţiunile

$$IRØ = MAX(IRØ,\ IR1)$$
$$IR1 = MIN(IRØ,\ IR1)$$

şi apoi continuăm subprogramul ca mai sus.

Sînt necesare unele precizări privind folosirea masivelor în funcţia subprogram. Orice masiv trebuie declarat la începutul programului. (Toate celelalte declarări trebuie să se facă aici). Am văzut însă că declarările masivelor trebuie să se facă precizînd exact numărul elementelor masivului; în subprogramul *FUNCTION* acest lucru nu este necesar şi din această cauză declararea poate fi de una din formele

$$DIMENSION\ A(1)$$
$$DIMENSION\ A(N)$$

Nu este deci necesar să se precizeze în subprogram numărul de elemente ale masivului; subprogramul informează numai compilatorul că unul din argumentele sale formale este un masiv. Totuşi, în programul în care se foloseşte masivul, acesta trebuie declarat din nou, precizîndu-se numărul său maxim de elemente.

O funcţie subprogram se poate referi la altă funcţie subprogram sau subrutină cu condiţia ca aceasta din urmă să fie definită în afara subprogramului.

3.9.3. **Subrutina** este un subprogram asemănător lui *FUNCTION*; ea este formată din instrucţiunea *SUBROUTINE* urmată de instruc-

Scanned with OKEN Scanner

țiuni FORTRAN, printre care există cel puțin un *RETURN* și un singur *END* la sfîrșitul subprogramului.
Instrucțiunea SUBROUTINE este de forma

SUBROUTINE nume (listă de argumente)

Se vede că forma acestei instrucțiuni este asemănătoare instrucțiunii *FUNCTION*; se pot spune aceleași lucruri și aici privind specificațiile, „nume", „listă de argumente". Deosebirea între *FUNCTION* și *SUBROUTINE* constă în faptul că *FUNCTION trebuie* să întoarcă programului principal (cînd este chemat de acesta) o valoare (rezultat), pe cînd *SUBROUTINE* nu întoarce neapărat programului nici un rezultat sau poate întoarce mai multe. (Din această cauză argumentele ei pot fi clasificate și ca *inițiale* și *rezultate*.)

Mai există de asemenea și alte deosebiri. Astfel, programul principal apelează o subrutină printr-o instrucțiune *CALL*; instrucțiunea imediat următoare lui *CALL* din programul principal se execută după ce s-a executat instrucțiunea *RETURN* a subrutinei. Și aici ca și în cazul *FUNCTION* se face distincție între *argumente formale* și *argumente curente*, iar în cazul cînd subrutina este chemată de rutina principală, trebuie să existe o corespondență ca tip, ordine, număr, între parametrii formali și cei actuali. Cele spuse în cazul *FUNCTION* referitor la masive se pot repeta identic și aici.

Subrutinele și funcțiile subprogram se pot referi una la alta; rezultatele unei subrutine sau funcții pot constitui valori ale argumentelor actuale pentru altă subrutină.

O subrutină poate chema alte subrutine, dar nu se poate chema pe ea însăși.

Orice nume de variabile, diferite de argumente, care apar într-un subprogram sînt „locale" acelui subprogram. Așa cum precizam și în cazul *FUNCTION*, numerele de instrucțiune într-un subprogram sînt de asemenea „locale"; nu ne putem referi la un număr de instrucțiune al unui subprogram printr-o instrucțiune din afara subprogramului.

Pentru a preciza modul de folosire a subrutinelor și a funcțiilor vom prezenta un exemplu.

Exemplul 3. Se dă o diviziune $X_1 \ldots X_{n+1}$ a intervalului (T_1, T_2) și $C_1, \ldots, C_n$ niște constante reale. Se presupun cunoscute constantele μ și σ. Se cere să se rezolve ecuația transcendentă în Δ

$$(1) \qquad S(\Delta) = \sum_{i=1}^{n} C_i \left(\exp\left(+ - \frac{1}{2} \frac{X_{i+1} + \Delta - \mu}{\sigma} \right)^2 \cdot \right.$$

$$\left. \cdot - \exp\left(- \frac{1}{2} \left(\frac{X_i + \Delta - \mu}{\sigma} \right)^2 \right) \right) = 0.$$

Scanned with OKEN Scanner

Vom construi două subrutine și anume:

a) O subrutină care să calculeze suma $S(\Delta)$.

b) O subrutină care rezolvă ecuația transcendentă prin metoda bisecțiunii intervalului, pe un interval (da, db). Această metodă constă în următoarele.

Se calculează produsul $P = S(da) \cdot S(db)$. Dacă $P < 0$, ecuația are o soluție în intervalul (da, db); se divide acest interval în două părți egale prin punctul $C = \frac{da + db}{2}$. Se calculează de asemenea $P_1 = S(da)\ S(c)$, $P_2 = S(c)\ S(db)$; rădăcina ecuației se va găsi în (da,c) sau (c, dp), după cum $P_1 < 0$ sau $P_2 < 0$. Dacă de exemplu, $P_1 < 0$, atunci se ia $db = c$ și se repetă procedeul de la început, pînă cînd $|dp - da| < \varepsilon$, unde ε este o eroare fixată. În acest caz, soluția va fi $c = \frac{da + db}{2}$. Dacă pe parcurs s-a găsit $P_1 = 0$ sau $P_2 = 0$, atunci soluția va fi tot c.

Dacă de la început $P > 0$, înseamnă că ori ecuația nu are rădăcină în (da, db), ori există mai multe rădăcini în (da, db), caz în care trebuie cercetate subintervale ale lui (da, db) pentru care $P < 0$.

Subrutina *SUM* (vezi Exemple de subrutine FORTRAN) calculează $S(\Delta)$. Parametrii formali ai subrutinei sînt:

— masivul C, care reprezintă vectorul $C_1, \ldots, C_n$;
— masivul X, care reprezintă vectorul $X_1, \ldots, X_{n+1}$;
— constanta AV, care reprezintă pe μ;
— constanta S, care reprezintă pe σ;
— constanta D, care reprezintă pe Δ;
— constanta N, care reprezintă numărul n al intervalelor;
— parametrul *SUMA*, care reprezintă valoarea sumei (singurul parametru formal de ieșire).

Exemple de subrutine FORTRAN

```
      SUBROUTINE SUM (C, X, AV, S, D, N, SUMA)
C     C ARE DIMENSIUNEA N IAR X ARE DIMENSIUNEA N + 1.
      DIMENSION C(1), X(1)
      SUMA = 0
      DO 1 I = 1, N
      A = 0.5*(X(I + 1) + D - AV)*(X(I + 1) + D - AV)/(S*S)
      B = 0.5*(X(I) + D - AV)*(X(I) + D - AV)/(S*S)
      T = C(1)*(E × P(-A) - EXP(-B))
      SUMA = SUMA + T
```


Scanned with OKEN Scanner

```
    1 CONTINUE
      RETURN
      END
      SUBROUTINE ECTRNS(C, X, AV, S, D, DA, DB, N, E, IER)
      DIMENSION C(1), X(1)
      IER = 0
      CALL SUM(C, X, AV, S, DB, N, FB)
   10 CALL SUM(C, X, AV, S, DA, N, FA)
      GO TO 2
    1 CALL SUM (C, X, AV, S, DB, N, FB)
    2 DC = (DA + DB)/2
      CALL SUM (C, X, AV, S, DC, N, FC)
      IF (FC*FB)3, 30, 4
    3 DA = DC
      IF (ABS(DA—DB) — E)20, 1, 1
    4 IF(FC*FA) 5, 30, 50
    5 DB = DC
      IF(ABS(DA—DB) — E)20, 10, 10
   50 IER = 1
   30 D = DC
      GO TO 40
   20 D = (DA + DB)/2
   40 RETURN
      END
      SUBROUTINE COST (C, X, AV, S, D, N, GAMA)
      DIMENSION C(1), X(1)
C     COST CALCULEAZA FUNCTIA COSTULUI MEDIU GAMA
      GAMA = 0.
      Y1 = (X(1) + D — AV)/S
      CALL NDTR(Y1, P1, D1)
      DO 1 I = 1, N
      Y2 = (X(I + 1) + D — AV)/S
      CALL NDTR(Y2, P2, D2)
      TERM = C(I)*(P2 — P1)
      GAMA = GAMA + TERM
      P1 = P2
    1 CONTINUE
      RETURN
      END
```

Subrutina *ECTRNS* calculează soluţia ecuaţiei. Parametrii săi formali sînt:

— masivele C şi X, constantele AV, S, limitele intervalului DA şi DB în care se caută soluţia parametrului N (cu semnificaţia de mai sus)

Scanned with OKEN Scanner

și constanta E care reprezintă eroarea (impusă) de determinarea soluției ; toți aceștia sînt parametri de intrare ;

— constanta D care este parametru de *ieșire* și reprezintă soluția $D \in (DA, DB)$;

— constanta IER care ia valorile 0 și 1 după cum ecuația are sau nu soluție în intervalul (DA, DB).

Exemplul de mai sus arată cum două subrutine se pot apela una pe alta.

Este de remarcat faptul că subrutina SUM poate fi înlocuită cu un subprogram de tip $FUNCTION$, care să nu conțină ca parametru $SUMA$ dar care, înainte de instrucțiunea $RETURN$, să aibă instrucțiunea

$$SUM = SUMA.$$

În încheiere vom prezenta un program principal al unei probleme care a condus la rezolvarea ecuației transcendente (1) din exemplul 3.

*Exemplul 4**. Problema de care ne ocupăm se ridică în studiul calității producției. Să presupunem că o caracteristică de calitate X are o repartiție normală $N(\mu, \sigma)$ și fie (T_1, T_2) intervalul său de toleranță. Să considerăm (ca în exemplul 3) o diviziune a intervalului $X_1 < \ldots < X_{n+1}$ și să notăm cu c_i costul provocat de faptul că $X_i \in (X_i, X_{i+1})$, $1 \leqslant i \leqslant n$. c_i poate lua valori negative. Presupunem că μ este mijlocul intervalului de toleranță și fie Δ deplasarea *centrului de grupare* (a reglajului) față de μ. Deci valoarea medie a lui X este $E(X) = \mu - \Delta$. Probabilitatea ca X să ia valori în intervalul (X_i, X_{i+1}) este

$$p(\Delta) = \frac{1}{\sigma\sqrt{2\pi}} \int_{X_i}^{X_{i+1}} \exp\left(-\frac{(x-\mu+\Delta)^2}{2\sigma^2}\right) dx, \tag{2}$$

iar costul mediu asociat caracteristicii de calitate X este

$$\Gamma(\Delta) = \sum_{i=1}^{n} c_i p_i(\Delta). \tag{3}$$

Dacă impunem condiția ca $\Gamma(\Delta) = \min$ și ținem seama de (2), rezultă ecuația transcendentă (1), din care trebuie determinat Δ optim. Punctul optim de reglaj pentru caracteristica X va fi $\mu - \Delta$.

---

* Pentru precizarea noțiunilor și notațiilor de statistică matematică și controlul calității, a se vedea [7].

Scanned with OKEN Scanner

Determinarea lui Δ optim se face utilizînd subrutinele din exemplul 3 și programul principal (vezi fig. 15) care este conceput astfel.

Se împarte intervalul de toleranță în $2k$ intervale

$$k = \frac{1}{2}\left[\frac{T_2 - T_1}{\sigma}\right] + 1,$$

unde pentru un număr real u, $[u]$ înseamnă „cel mai mare întreg mai mic sau egal cu u". Pornindu-se apoi de la stînga la dreapta, se consideră pe rînd intervalele care au extremitatea stîngă în unul din cele $2k$ puncte de diviziune iar lungimea $\frac{\sigma}{4}$. Pe fiecare astfel de interval, se rezolvă ecuația și se găsește un Δ. Apoi cu subrutina *COST* se determină valoarea costului mediu conform formulei (3); programul mai determină și un factor de eficiență de forma

$$e = \frac{\Gamma(\Delta)}{\Gamma(0)}. \tag{4}$$

Procedeul de „măturare" de la stînga la dreapta a intervalului de toleranță permite găsirea tuturor valorilor lui Δ care satisfac ecuația (1) a costurilor medii corespunzătoare, precum și a factorilor de eficiență. Aceste valori sînt tipărite în tabelul „Rezultatele căutării soluției". Pentru calculul costurilor medii cu subrutina *COST*, se folosește și subrutina *NDTR* din biblioteca de programe *SSP* („Scientific Subroutine Package") produse de firma IBM, subrutină care calculează funcția lui Laplace.

Fig. 15. Exemplu de program principal.

```
      DIMENSION C(10), X(11), D(50), FD(50), EF(50), L(50)
      READ (1, 20) N, T1, T2
   20 FORMAT (I5, 2F8.3)
      M = N + 1
      READ (1, 21)(C(I), I = 1, N), (X(I), I = 1, M), AV, S, E
   21 FORMAT (4F6.1/5F6.1/2F4.1, F6.3)
      WRITE (3, 6) (C(I), I = 1, 4), (X(J), J = 1, 5), AV, S, E
    6 FORMAT ('DATE INITIALE' /14('=')//' COSTURI = '20 × 4F8.1//
    1 STANDA'DIVIZ. INTERV. DE TOLERANTA = '5F7.1//' MEDIA =
      = 'F6.1//' DEV.
    2 RD = 'F4.1//' EROAREA ADMISA = E = 'F6.3///)
      T = T2 — T1
      U = T/S
      K = U
      K = K/2 + 1
      WRITE (3,50)K
```


Scanned with OKEN Scanner

```
 50 FORMAT (///' MARIMEA NORMATA A CIMPULUI DE TOLERANTA
    ='13 ///)
    CALL COST (C, X, AV, S, 0., N, F0)
    WRITE (3, 60) F0
 60 FORMAT (//' COSTUL LA REGLAJ NORMAL = 'F10.2///)
    IF(F0)101, 102, 101
101 CONTINUE
    IROR = 0
    IER = 0
    MAX I = 2*(4*K — 1)
    GO TO 103
102 F0 = 1.
    IROR = 1
    WRITE (3, 104) IROR
104 FORMAT ('DELTA = 0, IROR = 'I2////)
103 CONTINUE
    S2 = S/2
    S4 = S/4
    DA = — FLOAT (K) * S = S4
    DB = DA + S2
    WRITE (3, 22)
 22 FORMAT (//' '5X' REZULTATELE CAUTARII SOLUTIEI D=DELTA'/'
    '44('=')/
  1 //' '80('=')/'* L(I)* IER* DA* DB*
  2 D(I) * FD(I)* EF(I)*'/''86('=')/)
    DO 100 I = 1, MAXI
    DA = DA + S4
    DB = DB + S4
    A = DA
    B = DB
    CALL SUM(C, X, AV, S, A, N, FA)
    IF (FA)1, 10, 1
  1 CALL SUM(C, X, AV, S, B, N, FB)
    IF(FB*FA)2, 11, 3
  1 CALL ECTRNS (C, X, AV, S, D(I), A, B, N, E, IER)
    GO TO 18
 10 D(I) = A
    GO TO 18
 11 D(I) = B
 18 L(I) = I
    CALL COST (C, X, AV, S, D(I), N, FD(I))
    GO TO 17
```


Scanned with OKEN Scanner

```
  3 L(I) = 0
    D(I) = 1000.*S
    FD(I) = 0.
 17 EF(I) = FD(I)/F0
    WRITE (3, 15) L(I), IER, DA, DB, D(I), FD(I), EF(I)
 15 FORMAT(18, 9×12, 5×5(F10.3,2×)//)
100 CONTINUE
    WRITE (3, 36)
 36 FORMAT (' ', 85(' — ')//)
    A0 = FD(1)
    A1 = FD(1)
    L0 = 1
    L1 = 1
    DO 30 I = 1, MAXI
    IF(A0 = FD(I))31, 32, 32
 32 A0 = FD(I)
    L0 = I
 31 CONTINUE
    IF(AI = FD(I))33, 33, 34
 33 A1 = FD(I)
    L1 = I
 34 CONTINUE
 30 CONTINUE
    D0 = D(L0)
    D1 = D(L1)
    WRITE (3, 35) D0, A0, D1, A1
 35 FORMAT (//' PUNCTUL DE MINIM ABSOLUT = D0 = 'F10.5/ 'COS-
    TUL MINIM = AD = '
    1F12.5//' PUNCTUL DE MAXIM = D1 = 'F10.5/' COSTUL MAXIM =
    A1 = 'F12.5//)
    STOP
    END
```

Pentru a fi executat pe calculator, programul trebuie să conţină încă o serie de instrucţiuni, scrise începînd cu coloana 1; în final, el trebuie să se prezinte astfel:

```
// JOB 11102 ECTRANSC
// OPTION LINK
// ASSGN SYSRLB, X' xxx'
// EXEC FORTRAN
Subrutina 1
/*
```


Scanned with OKEN Scanner

```
// EXEC FORTRAN
Subrutina 2
/*
// EXEC FORTRAN
Subrutina 3
/*
// EXEC FORTRAN
Program principal
/*
// EXEC LNKEDT
// EXEC
Date* (μ, σ, X_i, 1 ⩽ i ⩽ n + 1, c_j, 1 ⩽ j ⩽ n)
/*
/
```

Pregătit sub această formă, programul a fost executat pe sistemul IBM 360/30 de la Centrul de Calcul al Universității București.

Limbajul FORTRAN dispune de multe alte facilități de programare, legate în special de memorarea datelor și de construcția fișierelor de date pe discuri și benzi magnetice, precum și de utilizarea acestora.

Exerciții

1. $10n$ numere reale sînt înregistrate pe n cartele cu formatul 1ØF8.3; toate numerele se găsesc în intervalul $[a, b]$. Se dă diviziunea $a = a_0 < a_1 < \ldots < a_{m-1} < a_m = b$ a intervalului $[a, b]$. Să se scrie o subrutină care să citească cele 10 n numere și să determine frecvențele lor pe intervalele $[a_{i-1}, a_i]$, $1 \leqslant i \leqslant m$.
2. Să se scrie un sistem de subrutine pentru calculul numerelor $n!$, A_n^k și C_n^k.
3. Pe n cartele sînt înregistrate $10n$ numere cu formatul 1ØF8.3. Fiecare cartelă conține observații asupra unui vector 10-dimensional ($n > 10$). Să se scrie o subrutină care să citească cele n numere și să calculeze matricea lor de corelație.
4. Fiind dată matricea A pozitiv definită, să se scrie o subrutină pentru calculul inversei sale prin metoda Gauss-Jordan [vezi 8]).
5. Să se scrie un program de testare a subrutinelor anterioare.

---

* Cartelele cu date se pregătesc în concordanță cu instrucțiunile format pentru citire.

Scanned with OKEN Scanner

Notă. Să plecăm de la sistemul

(1) $$y = Ax.$$

Dacă $a_{11} \neq 0$ (există un pivot pozitiv), atunci prima ecuație din (1) se poate rezolva în raport cu x_1 care se va introduce în celelalte ecuații. Deci (1) conduce la

(2) $$\begin{aligned} x_1 &= a'_{11}y_1 + a'_{12}x_2 + \ldots + a'_{1n}x_n, \\ y_2 &= a'_{21}y_1 + a'_{22}x_2 + \ldots + a'_{2n}x_n, \\ &\vdots \\ y_n &= a'_{n1}y_1 + a'_{n2}x_2 + \ldots + a'_{nn}x_n, \end{aligned}$$

unde

(3) $$a'_{11} = \frac{1}{a_{11}}, \quad a'_{1j} = -\frac{a_{1j}}{a_{11}}, \quad a'_{i1} = \frac{a_{i1}}{a_{11}},$$

$$a'_{ij} = a_{ij} - \frac{a_{i1}a_{ij}}{a_{11}} \quad (i, j = 2, \ldots, n);$$

Să presupunem acum că $a'_{22} \neq 0$; se rezolvă cea de-a doua ecuație din (2), în raport cu x_2, și se înlocuiește în celelalte ecuații. Repetînd procedeul, se ajunge în final la

(4) $$x = By.$$

Deci $B = A^{-1}$, adică în n pași de tipul (3) se obține inversa. Formulele (3) definesc în fapt metoda Gauss-Jordan de inversare a matricei.

Condiția ca matricea A să fie pozitiv definită asigură la fiecare pas existența *pivotului* (a_{11}, a'_{22}, etc.) pozitiv.

Se poate constata că este necesar să se calculeze numai elementele inversei de sub diagonala principală $\left(a_{ij}^{-1},\ j > i\right)$, întrucît cele de deasupra se obțin prin transpunere și anume

(5) $$a_{ji}^{-1} = -a_{ij}^{-1}, \quad j < i.$$

3.10. RĂSPUNSURI LA EXERCIȚII

§ 3.3.

1. *INX*, 1, pentru că conține virgulă;
AM/3, pentru că folosește /;
3IJJ, pentru că începe cu o cifră;
I4567Ø0, pentru că are 7 caractere;
X2Y3Z45, pentru că are 7 caractere.

2. Prima instrucțiune are următoarele erori:

a) *DIMENSION*, trebuie scris *DIMENSION* (fără virgulă).

Scanned with OKEN Scanner

I47(21, 54, 75, 2) nu este un masiv—FORTRAN, deoarece are patru indici; el poate fi eventual declarat cu ajutorul a doi masivi; I471(21, 54, 75), I472(21, 54, 75).

În aceeași declarare, înainte de X și Y trebuie incluse virgule.

b) Înainte de B trebuie virgulă,

c) Masivul LOG apare și ca întreg și ca logic.

d) Lipsește virgula între $C1$ și $C2$.

e) Nu are nici o eroare.

3. 1547. (conține un punct);
± 131 (are semn dublu).

4. 3,15 (conține virgula);
$17\,E+02$ (mantisa nu e scrisă corect).

5. (2,3.) (cele două constante nu au același tip);
(Ø. 3 $E+1$Ø, Ø.5D —Ø1) (cele două constante nu au același tip).

6. Variabile întregi: $INTR$, LAC, $NORA$;
Variabile reale: $TIMP$, $AMAX$, $XNORMA$.

§ **3.4.**

1. $(5*X**7 + (4.*X + 3.)**2)/(X*X - X + 1.)$

2. Expresia ia valoarea 15.
Dacă J este de tip real, atunci $I/J = 1.5$ și deci expresia ia valoarea 15.5.

3. Expresia corectată devine
$A * B + (-X)*Y + 7.5 * (A**X + Y)$

4. Valoarea expresiei este $.TRUE.$

5. Ordinea operațiilor este $.GT.$, $.NOT.$, $.AND.$, $.OR.$
Valoarea expresiei este $.FALSE.$

§ **3.5.**

1. $(SIN(X**3 - X + 1.))**2 + EXP(-3.*X) - COS(2.*X - 3) + SQRT(EXP(5.*X + 1))*ALOG(X * X + 5.)$

2. $2.*COS(X) + 3.*TAN(2.*X) + INT(TAN(ALOG(X * X + 3.)))*SIN(ALOG(X))$

3. $X = MOD(A*Y, M)$(A, X, Y, M — întregi);
$X = AMOD(A*Y, M)$(A, X, Y, M — reali).

Scanned with OKEN Scanner

§ **3.6.**

1.
```
      IF(X - 5.)1, 2, 3
1     F = X * X + X + 1.
      GO TO 4
2     7.*X - 4
      GO TO 4
3     2.**X - 1
4     CONTINUE
```

2.
```
      PI = 3.14159273
      A = ABS(X)
      IF(A - PI/2.)2,3,3
2     G = 2.*X* COS(X)
      GO TO 5
3     IF((P/2.).LE.A.AND.A.LT.PI) GO TO 4
      G = X * EXP(- X)
      GO TO 5
4     G = TAN(X/2.) + EXP(X)
5     F = SIGN(X) * G
```

3. Nu se execută instrucţiunile cu etichetele 4 şi 6.
$Y = 27.$

§ **3.7.**

1. Instrucţiunea *IF* transferă controlul în interiorul buclei (instrucţiunea 8).

2. Prima buclă *DO* transferă controlul la instrucţiunea 1Ø care se află în interiorul celei de-a doua bucle.

3. Trebuie făcută mai întîi declararea masivului; instrucţiunile sînt
```
      DIMENSION A(3Ø, 3Ø)
      DO 1 I = 1,3Ø
      DO 1 J = 1,3Ø
      X = A(I, J)
      A(I, J) = A(J, I)
1     A(J, I) = X
```

4. Instrucţiunile, împreună cu declarările, sînt ($N \leqslant 1ØØ$)
```
      DIMENSION X (1ØØ)
      DOUBLE PRECISION S
      XBAR =Ø.
      S =Ø.
```


Scanned with OKEN Scanner

```
      DO 1 I = 1, N
      XBAR = XBAR + X(I)
    1 S = S + X(I) * X(I)
      XBAR = XBAR/N
      S = S/N - XBAR * XBAR
      S = DSQRT(S)
```

§ **3.8.**

1. a) $N = 12$ $A = 3Ø.4$ $B = 531.25$ $C = 47.1$ $D = 234.56$

b) Prima instrucțiune nu este corectă deoarece, datorită semnului //, fișierul se termină înainte de a se citi valoarea lui C.
Cea de-a doua este corectă și introduce în memorie datele

$$M = 123 \qquad X = 0.453 \qquad XX = 1.25Ø$$

2. Rezultatul tipăririi începînd cu prima coloană este 2Ø145.2351.31
23.2

3. Nu sînt.
Instrucțiunea cu numărul 3 trebuie să aibă codul H scris ca $4H$ *VOL*.
Instrucțiunea cu numărul 5 necesită o paranteză la sfîrșit. Instrucțiunea 6 poate fi acceptată pentru ieșire dar nu pentru intrare (înregistrarea are 124 caractere, pe cînd cartela poate avea maximum 8Ø).

4. a) Rezultatul tipăririi, începînd din prima coloană, este

— *MEDIA* = — — — — — 32.5
— *DISPERSIA* = — — — 7.6
unde liniuțele reprezintă blancuri.

b) Rezultatul este
— *DELTA ESTE* * * * * * *
— — — — — — — — — — — — —
— — — — — — — — — — — — —
— — N = — — F
unde liniuțele reprezintă de asemenea blancuri.

§ **3.9.**

1. Subrutina folosește masivele : X de dimensiune N pentru memorarea vectorului de pe cartelă, F de dimensiune M pentru memorarea frecvențelor și A de dimensiune $M + 1$ pentru memorarea diviziunii. Programul este :

Scanned with OKEN Scanner

```
      SUBROUTINE FRECV(X, F, A, M, N)
      DIMENSION X(1), A(1), F(1)
C     ELEMENTELE LUI A SE PRESUPUN MEMORATE
      DO 1 I = 1, M
    1 F(I) = O.O.
      DO 4 J = 1, N
      READ (1, 3) (X(I), I = 1, 10)
    3 FORMAT (10F 8.3)
      DO 4 I = 1, 10
      DO 4 K = 1, M
      IF(X(T) - A(K + 1))4, 4, 5
    5 F(K) = F(K) + 1.0
    4 CONTINUE
      RETURN
      END
```

2. Subrutinele, sînt următoarele:

a) Pentru $n!$

```
      SUBROUTINE FACT (N, F)
      F = 1.0
      DO 1 I = 1, N
    1 F = F*I
      RETURN
      END
```

b) Pentru $A_n^k = \frac{n!}{k!}$

```
      SUBROUTINE ARANJ(N, K, A)
      CALL FACT (N, F1)
      CALL FACT (K, F2)
      A = F1/F2
      RETURN
      END
```

c) Pentru $C_n^k = \frac{n!}{k!(n-k!)}$

```
      SUBROUTINE COMB (N, K, C)
      CALL FACT (N, F1)
      CALL FACT (K, F2)
      K3 = N - K
      CALL FACT (K3, F3)
      C = F1/(F2*F3)
      RETURN
      END
```


Scanned with OKEN Scanner

3. Matricea de corelaţie este o matrice simetrică ale cărei elemente sînt definite de relaţia

$$r_{ij} = \frac{\frac{1}{n}\sum_{\alpha=1}^{n}(X_{i\alpha} - \bar{X}_i)(X_{j\alpha} - \bar{X}_j)}{s_i\, s_j};$$

$$\bar{X}_k = \frac{1}{n}\sum_{\alpha=1}^{n} X_{k\alpha}, \quad s_k^2 = \frac{1}{n}\sum_{\alpha=1}^{n}(X_{k\alpha} - \bar{X}_k)^2, \quad 1 \leqslant k \leqslant 10,$$

unde $(X_{1\alpha}, \ldots, X_{k\alpha})'$, $1 \leqslant \alpha \leqslant n$ $(k = 10)$ sînt observaţiile asupra vectorului $(X_1, \ldots, X_k)$.
Din formulele precedente se deduce formula

$$r_{ij} = \frac{n\sum_{\alpha=1}^{n} X_{i\alpha}X_{j\alpha} - \left(\sum_{\alpha=1}^{n} X_{i\alpha}\right)\left(\sum_{\alpha=1}^{n} X_{j\alpha}\right)}{\sqrt{\left[n\sum_{\alpha=1}^{n} X_{i\alpha}^2 - \left(\sum_{\alpha=1}^{n} X_{i\alpha}\right)^2\right]\left[n\sum_{\alpha=1}^{n} X_{j\alpha}^2 - \left(\sum_{\alpha=1}^{n} X_{j\alpha}\right)^2\right]}}$$

care este mult mai simplu de calculat.
Subrutina care calculează matricea $R = (r_{ij})$, conform formulei precedente, este următoarea:

```
      SUBROUTINE COR (N, X, XBAR, S, R)
      DIMENSION X(1), XBAR (1), R(1, 1)
      DOUBLE PRECISION S(1)
      DO 1 I = 1, 10
      S(I) = 0.0
      XBAR (I) = 0.0
      DO 1 J = 1, 10
    1 R(I, J) = 0.0
      DO 2 L = 1, N
      READ (1, 3)(X(I), I = 1, 10)
    3 FORMAT (10F8.3)
      DO 2 I = 1.10
      S(I) = S(I) + X(I)*X(I)
      XBAR (I) = XBAR(I) + X(I)
      DO 2 J = 1, I
    2 R(I, J) = R(I, J) + X(I)*X(J)
      DO 4 I = 1, 10
      S(I) = S(I)*N — XBAR(I)*XBAR(I)
    4 S(I) = DSORT(S(I))
      DO 5 I = 1, 10
      DO 5 J = 1, I
```


Scanned with OKEN Scanner

```
      R(I, J) = R(I, J)*N - XBAR(I)*XBAR(J)
    5 R(I, J) = R(I, J)/(S(I)*S(J))
      DO 6 J = 1, 10
      DO 6 I = 1, J
    6 R(I, J) = R(J, I)
C     ULTIMUL CICLU DETERMINA TRIUNGHIUL SUPERIOR AL MATRICEI
      RETURN
      END
```

Observație (1). Dacă în subrutină se introduce un nou parametru formal k pentru a desemna dimensiunea vectorului X, atunci instrucțiunea READ trebuie schimbată în mod corespunzător pentru a citi k date.

(2). Instrucțiunea *DOUBLE PRECISION* a fost utilizată pentru a se evita eventualele erori în calculul lui $S(I)$, care ar fi putut face ca acestea să fie negative.

(3) Pentru valori mari ale lui N, este posibil ca la calculul lui $S(I)$ sau $R(I, J)$, $1 \leqslant I, J \leqslant 10$ să se obțină depășire de registru. Pentru a se evita acest lucru este necesar să se considere în loc de $X(I)$, abaterile $X(I) - T(I)$. Valorile $T(I)$ se obțin prin memorarea a $K = 10$ valori ale vectorului, calculîndu-se $T(I)$ ca medii aritmetice. Un astfel de program este prea lung pentru a fi prezentat aici. Cititorul interesat poate consulta documentația IBM, referitoare la *SSP*.

Alegerea valorilor $T(I)$ se poate face încă de la început de către utilizator, înregistrîndu-se pe cartele direct abaterile $X(I) - T(I)$.

4. Subrutina este

```
      SUBROUTINE GAUJ (A, N, IER, H)
      DIMENSION A(1. 1), H(1)
      IER = 0
      K = N + 1
    1 K = K - 1
      P = A(1, 1)
      IF(P)10, 10, 2
    2 CONTINUE
      DO 3 I = 2, N
      Q = A(I, 1)
      IF(I - K)5, 5, 6
    5 H(1) = -Q
      GO TO 7
    6 H(I) = Q
    7 H(I) = H(I)/P
      DO 3 J = 2, I
      A(I - 1, J - 1) = A(I, J) + Q*H(J)
    3 CONTINUE
      A(N, N) = 1.0/P
      DO 8 I = 2, N
    8 A(N, I - 1) = H(I)
      IF(K - 1)9, 9, 1
```


Scanned with OKEN Scanner

```
   10 IER = 1
      WRITE (3, 12)
   12 FORMAT ('EROARE')
    9 CONTINUE
      RETURN
      END
```

Observație. Subrutina transformă numai partea de sub diagonala principală a matricei *A*. Pentru a afla și partea de deasupra diagonalei, trebuie adăugate instrucțiunile care realizează (5).

5. Programul este următorul:

```
      DIMENSION XX(10), F(4), A(5), X(10), XBAR (10), R(10, 10), AA(5,
      5),H(5)
      DOUBLE PRECISION S(10)
      DATA A/73., 84., 12., 18., 2./
      CALL ARANJ (4, 1, SF)
      WRITE (3, 100) SF
  100 FORMAT (' ','ARANJAMENTE', F11.1)
      CALL FRECV(XX, F, A, 4, 3)
      WRITE (3, 1)F
    1 FORMAT (' ', 'F ESTE', 4 (F10.0,5X))
      CALL COMB (10, 3, C)
      WRITE (3, 2)C
    2 FORMAT (' ', 'COMBINARI DE 10 LUATE CITE 3', E 15.8)
      CALL COR (50, X, XBAR, S, R)
      WRITE (3, 9)X
    9 FORMAT (' ', 'X ESTE', 5(E 15.8,3X))
      WRITE (3, 10)XBAR
   10 FORMAT (' ', 'XBAR ESTE', 5(E 15. 8,3 X))
      WRITE (3, 11)S
   11 FORMAT (' ', 'S ESTE', 3 (D15.8, 3X))
      WRITE (3, 12)R
   12 FORMAT (' ', 'R ESTE', 5(E15.8, 3X))
      CALL GAUJ (AA, 5, IER, H)
      DATA AA/2*7.8,3*9.23,4*8.41,11*3.14,5*5./
      WRITE (3, 15)AA
   15 FORMAT (' ', 'AA ESTE', 5 (E 15.8))
      STOP
      END
```


Scanned with OKEN Scanner

BIBLIOGRAFIE

[1] DREYFUS, M., *Fortran IV*, Paris, Dunond, 1967.

[2] McCRACKEN, D. D. and DORN, W. S., *Numerical Methods and Fortran programming*, New-York, 1964.

[3] IBM System/360, *Fortran IV Language.*

[4] IBM System/360, *Fortran IV Programmer's Guide.*

[5] IBM System/360 (360 A—CM—03X) — *Scientific Subroutine Packager, Programmer's Manual, Version III*, H20—0205—3

[6] Sofonea, L., Botea, A., *Elemente de programare automată. FORTRAN IV*, București, 1972.

[7] Rancu, N., Tövissi, L., *Statistica matematică cu aplicații în producție*, București Ed. Academiei, 1963.

[8] Wilkinson, J., Reinsch, C., *Handbook for Automatic Computation*, LINEAR ALGEBRA, Vol. II, Springer-Verlag, Berlin, 1971, pp. 45—46.

Scanned with OKEN Scanner

Capitolul IV

FOLOSIREA MEMORIILOR EXTERNE ÎN PROBLEMELE DE CERCETARE ȘTIINȚIFICĂ

4.1. ORGANIZAREA DATELOR UNEI PROBLEME

4.1.1. Fișierul și elementele regăsirii informației

În capitolul I am văzut că memoria principală conține programul care se execută și datele acestuia. În general nu este necesar ca toate datele pe care le prelucrează programul să se afle de la început în memorie (dacă acest lucru ar fi necesar, atunci s-ar putea ca datele să nu încapă în memoria principală). Ele pot fi citite, de exemplu, pe rînd din cititorul de cartele. Acest mod de lucru prezintă dezavantajul evident de a pune în mod continuu cartele în cititor. În afară de aceasta, se poate ca unele date deja citite și prelucrate să se distrugă în memorie deși ulterior mai este nevoie de ele. Atunci ele trebuie să fie citite din nou.

Din această prezentare sumară se vede că a apărut necesitatea folosirii unor memorii externe pe care se pot depune (teoretic) oricît de multe date, care nu se șterg decît dacă dorește utilizatorul. Memoriile externe la care ne referim sînt unitățile de discuri și unitățile de benzi. Datele problemei le numim *informații*.

Vom privi informația din două puncte de vedere:

1. fizic, adică al purtătorului fizic de informație;
2. logic, adică al legăturilor dintre datele care constituie informația.

Fizic, *volumul* este cea mai mare unitate de informație: un pachet de discuri, o rolă de bandă. Volumul se compune din *înregistrări fizice*, numite *blocuri*. Lungimea unui bloc este funcție de dorința utilizatorului. Ea reiese din lista de variabile care însoțește instrucțiunile *READ* și *WRITE* care se adresează discurilor și benzilor și anume este suma lungimilor tuturor variabilelor din listă. (Noțiunea de înregistrare fizică se întîlnește și la alte unități de intrare/ieșire diferite de discuri și benzi. De exemplu, pentru cititorul de cartele, o înregistrare fizică este o cartelă, iar pentru imprimantă este o linie întreagă).

Informațiile dintr-un volum pot să nu fie corelate între ele, adică pot fi destinate pentru programe distincte și utilizatori diferiți. *Informațiile corelate* între ele formează o *înregistrare logică*. O înregistrare fizică (un bloc) este formată din una sau mai multe înregistrări logice. Totalitatea înregistrărilor logice cu aceeași destinație și organizare

Scanned with OKEN Scanner

formează *fișierul*. Pe un volum se pot găsi unul sau mai multe fișiere, iar un fișier se poate întinde pe unul sau mai multe volume. Deci fișierul este cea mai mare unitate de informație din punct de vedere logic (în care datele sînt corelate între ele). El este format din blocuri, iar blocurile sînt formate din înregistrări logice. Atragem atenția că, pentru unitățile de discuri și de benzi, totalitatea variabilelor din lista întreagă de la instrucțiunile *READ* și *WRITE* (care se adresează memoriilor externe) formează o singură înregistrare fizică, deci un bloc. Înregistrările logice, din care este formată înregistrarea fizică, au importanță numai pentru utilizator, pentru că prin ele utilizatorul știe destinația pentru programul său a fiecărei „bucăți" din bloc.

Pentru exemplificare, fie problema transporturilor. Aici intervin patru tipuri diferite de informații, care vor conduce la patru fișiere: fișierul cantităților solicitate, fișierul cantităților furnizorilor, fișierul distanțelor furnizor — beneficiar și fișierul costurilor de transport de la furnizor la beneficiar.

La organizarea informației se pun problemele înmagazinării ei cu siguranță în păstrare-utilizare și cu regăsirea rapidă a informației dorite. Pentru aceasta, la informația propriu-zisă, pe care utilizatrul dorește să o memoreze pe disc sau pe bandă, se mai adaugă o informație despre această informație, pe care o vom numi *meta-informația fișierului* sau *informația de informare*. Metainformația este dată de utilizator prin programul sursă. Memorarea ei se face într-un cîmp special de pe disc sau de pe bandă, numai prin acțiunea sistemului de operare, fără intervenția utilizatorului. (La IBM 360 metainformația fișierului se dă prin cartelele prelucrate de programul Job Control). Cîmpurile pentru metainformații împreună cu conținuturile lor formează *etichetele*. În general se folosesc etichete cu formă standard impusă de sistemul de operare (pe care utilizatorul nu este obligat să o cunoască).

Pentru găsirea unei informații (care constituie datele problemei) este necesară identificarea volumului care o conține și a fișierului în care se află. Aceasta se face cu ajutorul etichetelor.

Fiecare volum (pachet de discuri sau rolă de bandă) are un anume cod de acces la conținutul lui format din șase caractere. Codul se numește *seria volumului* și este pus de utilizator, după dorința sa, în operația de *inițializare a volumului*. De obicei, într-un centru de calcul, toate pachetele de discuri și rolele de bandă sînt inițializate de operator, iar seriile stabilite sînt comunicate utilizatorului cînd acesta solicită discuri sau benzi. Evident, utilizatorul poate modifica seria prin înscrierea unui alt cod care devine cifru personal. (Detalii asupra etichetelor se pot găsi în [1], [2]).

Scanned with OKEN Scanner

4.1.2. Forma înregistrărilor logice și organizarea conținutului fișierului

Ne ocupăm de memorarea informațiilor pe disc sau pe bandă, făcînd distincție numai acolo unde este cazul. Modul de lucru standard permite memorarea informațiilor în una din următoarele forme:

1. *Înregistrări logice de lungime fixă, negrupate.* În acest caz, o înregistrare logică R = o înregistrare fizică R = un bloc B. Toate blocurile au deci aceeași lungime L. Blocurile sînt separate între ele prin niște goluri. Golurile ocupă o porțiune destul de mare pe pista discului sau pe bandă. De exemplu, la unele tipuri de benzi, golul este de 1,5 cm, chiar dacă blocul ocupă fracțiuni de centimetru. Acesta este un spațiu de-a lungul căruia banda atinge viteza de regim, pentru citire/scriere. Indicele de utilizare a volumului este cu atît mai mic cu cît L este mai mic și cu cît numărul blocurilor este mai mare.

2. *Înregistrări logice de lungime fixă, grupate.* Mai multe înregistrări logice R, fiecare de lungime L, formează un bloc B. Numărul înregistrărilor logice dintr-un bloc se numește *factor de grupare* sau *factor de blocare*. Acest factor este la dispoziția utilizatorului. Evident, indicele de folosire a volumului a crescut, comparativ cu înregistrările negrupate, prin eliminarea golurilor dintre înregistrările ce formează blocul. Se mențin, ca absolut obligatorii, golurile dintre blocurile unui fișier.

3. *Înregistrări logice de lungime variabilă, negrupate.* La lungimea înregistrării propriu-zise a utilizatorului, se mai adaugă automat un cîmp în care se reține lungimea înregistrării și lungimea blocului în care se află înregistrarea. Deci, blocul este format din înregistrarea logică la care se adaugă cîmpul de memorare a lungimilor. Blocurile sînt despărțite prin goluri.

4. *Înregistrări logice de lungime variabilă, grupate.* Blocul este format dintr-un număr (factorul de grupare) de înregistrări logice, la care se adaugă cîmpurile de memorare a lungimilor fiecărei înregistrări și a lungimii blocului însuși.

5. *Înregistrări logice nedefinite*

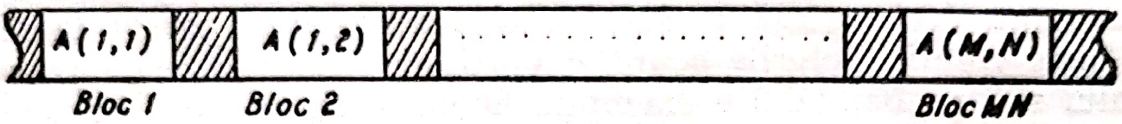
Cele mai des folosite sînt primele patru forme de înregistrare.

Pe un volum, informația și metainformația se depun în blocuri, despărțite prin goluri. Rolul golurilor este și acela de a indica sistemului de citire-scriere, de unde pînă unde se întinde un bloc, pentru că pe un volum sau de pe un volum se transferă numai blocuri.

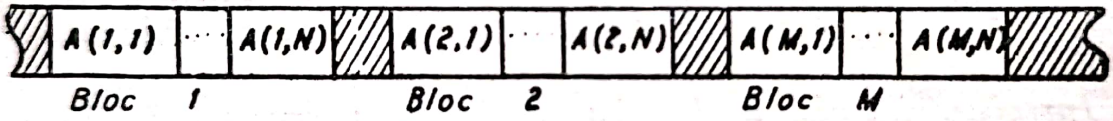
Scanned with OKEN Scanner

Dăm un exemplu de organizare a unui fișier cu elementele unei matrice A. Notăm elementele ei cu $A(I, J)$, $I = 1, \ldots, M$; $J = 1, \ldots, N$.

a) *Fișier cu înregistrări logice de lungime fixă, negrupate.* În acest, caz fiecare înregistrare logică $A(I, J)$ formează un bloc și deci fișierul are MN blocuri. În lista *READ* și *WRITE* va apare doar variabila $A(I, J)$, unde I și J au valori determinate. Pe pistă fișierul arată astfel:



b) *Fișier cu înregistrări logice de lungime fixă, grupate.* Grupăm înregistrările logice pe linie, adică factorul de grupare este N. Atunci fiecare linie formează un bloc și deci fișierul este compus din M blocuri. Dacă I este variabila care numără blocurile, atunci lista de variabile în instrucțiunile *READ* și *WRITE* este $(A(I, J), J = 1, N)$. La prima folosire a lui *READ* sau *WRITE* $I = 1$, la următoarea folosire $I = 2$ și la ultima $I = M$. Pe pistă fișierul arată astfel:



Din acest fișier, pe care îl presupunem depus pe disc, nu este posibilă citirea numai a unui singur element al matricei A. Este necesară citirea blocului întreg și apoi din bloc se extrage elementul dorit. Despărțirea blocului în elementele sale constituiente se face după ce el a fost adus (citit din volum) în memoria principală.

Prin acest exemplu se vede că este obligatorie citirea informației dintr-un fișier, exact în forma în care ea a fost depusă la crearea fișierului.

Din punctul de vedere al regăsirii informației într-un volum, apare noțiunea de *organizare a fișierului*. Menționăm trei tipuri de organizare, cele mai des folosite:

1. Organizarea secvențială;
2. Organizarea indexat-secvențială;
3. Organizarea directă.

Fiecărui tip de organizare îi corespunde un mod propriu de prelucrare a fișierului.

Scanned with OKEN Scanner

La organizarea secvenţială informaţia este memorată, citită şi prelucrată secvenţial. Nu se pot şterge sau adăuga înregistrări dacă nu se copiază tot fişierul. Fişierele secvenţiale se pot organiza pe orice tip de suport de informaţie.

Organizarea indexat-secvenţială este o formă superioară de organizare secvenţială, în care înregistrările au asociate nişte indexuri care permit regăsirea mai rapidă a informaţiei căutate. Fişierele indexat-secvenţiale se pot crea numai pe disc, tambur sau folii magnetice. Tot numai cu aceste tipuri de memorii se pot folosi fişierele organizate direct. La aceste fişiere, fiecare înregistrare are o cheie cu ajutorul căreia se determină rapid adresa informaţiei căutate.

Regăsirea informaţiei într-un volum este legată de natura accesului pe care îl permite volumul. În acest sens, accesul este de două feluri: *acces secvenţial* şi *acces direct*. În paragrafele următoare ale acestui capitol ne vom ocupa separat de memoriile cu acces direct şi de memoriile cu acces secvenţial.

4.2. CREAREA ŞI PRELUCRAREA FIŞIERELOR DE PE DISCURI MAGNETICE

4.2.1. Unităţile de discuri-memorii cu acces direct. Organizarea şi prelucrarea fişierelor secvenţial şi direct

Memoriile cu acces direct sînt acele memorii externe care permit depunerea oricărei informaţii sau accesul la orice informaţie, indiferent de timp şi de locul din volum. În acest sens, unităţile de discuri sînt memorii cu acces direct. Accesul la o anumită înregistrare de pe disc se face prin indicarea de către utilizator, prin programul său, a cilindrului şi a pistei de unde începe fişierul ce conţine înregistrarea dorită şi prin indicarea numărului de ordine al înregistrării (al blocului) în fişier.

Pentru concretizarea celor prezentate pînă aici în acest capitol, ne vom referi numai la unităţile de discuri 2311, utilizabile la configuraţia descrisă în § 1.2.6. Apoi, dintre modurile de organizare ale unui fişier, vom trata organizarea secvenţială şi organizarea în acces direct.

Indiferent de organizare, se pun următoarele probleme:

a) cît spaţiu trebuie să se aloce pe disc pentru fişierul dorit;
b) cum se alocă acest spaţiu;
c) cum se realizează gruparea prin program a înregistrărilor logice;

Scanned with OKEN Scanner

d) cum se depune informaţia în fişier la organizarea secvenţială şi la organizarea directă ;

e) cum se citeşte informaţia din fişier la organizarea secvenţială şi la organizarea directă ;

f) la organizarea directă : cum se creează fişierul pentru regăsirea oricărei înregistrări. În cele ce urmează vom răspunde la toate aceste probleme.

Am văzut că la informaţia propriu-zisă a utilizatorului se mai adaugă metainformaţia de descriere a fişierului. Pe utilizator îl interesează doar spaţiul rezervat pe fiecare pistă pentru datele problemei sale. Vom numi *capacitate a pistei* spaţiul de memorare a informaţiei propriu-zise. Astfel, capacitatea pistei este de 3625 baiţi (*capacitatea totală* a pistei este mai mare cu 61 baiţi). Pentru o înregistrare fizică nu este permisă o lungime mai mare de 3625 baiţi. Tabelul 17 indică lungimea L a unei înregistrări fizice şi numărul (NP) de înregistrări fizice pe o pistă.

TABELUL 17

NP	L	NP	L	NP	L	NP	L	NP	L
1	3625	7	447	13	213	19	127	25	82
2	1740	8	384	14	193	20	118	26	77
3	1131	9	334	15	177	21	109	27	72
4	830	10	295	16	162	22	102	28	67
5	651	11	263	17	149	23	95	29	63
6	532	12	236	18	138	24	88	30	59

Pentru lungimile neincluse în tabel, dar cuprinse în limitele tabelului, se face interpolarea luînd ca referinţă lungimea imediat mai mare decît lungimea studiată. De exemplu, dacă $L = 100$, atunci se pot memora 22 de înregistrări pe o pistă. Pentru lungimi L necuprinse în limitele tabelului se foloseşte o formulă simplă de calculare a lui NP şi anume

$$NP = 1 + \frac{3625 - L}{LC},$$

Scanned with OKEN Scanner

unde

$$LC = 61 + \frac{537}{512} \times L.$$

Rezultatele se rotunjesc la primul întreg mai mare. De exemplu, pentru $L = 7$ se obține $NP = 54$.

La înregistrările grupate, L rezultă din suma lungimilor tuturor variabilelor din lista lui *READ* sau *WRITE*. De exemplu, dacă pentru fiecare variabilă sînt alocați patru baiți (cum se face în mod obișnuit în FORTRAN), atunci lista $(A(I), I = 1,15)$ implică lungimea $L = 60$ și deci $NP = 29$. Cunoscînd numărul de blocuri ale fișierului, se poate determina cîte piste va ocupa el pe disc.

Răspunsul la problema (b) se găsește în paragraful următor.

Gruparea prin program a înregistrărilor logice (c) se realizează prin descrierea tuturor variabilelor, pe care vrem să le grupăm, în lista de variabile a instrucțiunilor *READ* și *WRITE*. De exemplu, lista $A, B, C, (D(I), I = 1,10)$ realizează gruparea lui $A, B, C, D(1), \ldots D(10)$ într-o singură înregistrare fizică pe disc.

Pentru problemele (d) și (e) se răspunde diferențiat, funcție de organizarea fișierului.

La organizarea secvențială, instrucțiunea de scriere (depunere) pe disc este *WRITE* (*a*, *b*) *lista*. Parametrul *a* este un număr de referință al setului de informație care este prelucrat. El rezultă din tabelul 18 și anume este numărul care se află în stînga numelui simbolic al utilizatorului (numele simbolic fiind ales arbitrar de utilizator din coloana din mijloc). De exemplu, pentru *SYS*007 rezultă $a = 10$. Parametrul *b* este marca sau eticheta instrucțiunii *FORMAT*. El apare numai cînd scrierea pe disc a variabilelor din listă se face într-o anume formă dorită de utilizator. De cele mai multe ori scrierea pe disc se face fără ca utilizatorul să indice formatul de scriere. Atunci, în FORTRAN, fiecărei variabile din listă îi sînt rezervați patru baiți pe disc (dacă utilizatorul nu a indicat alte lungimi pentru variabile). Lista indică variabilele ale căror valori vor fi scrise pe disc.

Instrucțiunea de citire de pe disc la organizarea secvențială este *READ* (*a*, *b*) *listă*, unde *a* și *b* și lista au aceeași semnificație. Menționăm că dacă scrierea se face cu un anume format este obligatoriu ca citirea să se facă tot cu același format. De asemenea, este obligatoriu ca lista de variabile de la citirea de pe disc să fie identică cu lista de variabile de la scrierea pe disc.

Fie, de exemplu, pentru o problemă de transporturi, matricea *A* a distanțelor de ordinul *A* (20, 100). Scris fără format (deci *b* nu apare), acest fișier va ocupa $20 \times 100 \times 4 = 8000$ baiți. Depunem distanțele grupate pe linii. Vor exista deci 20 de înregistrări fizice.

Scanned with OKEN Scanner

Dacă alegem numele *SYS007*, atunci scrierea pe disc se realizează prin

```
DO 1 I = 1,20
1 WRITE (10)(A(I, J), J = 1,100)
```

Citirea de pe disc, adică aducerea în memoria principală a datelor depuse pe disc, se face *numai secvențial*. Deci pentru a avea acces, de exemplu la linia 9, se citesc toate celelalte linii (înregistrări fizice) anterioare. Aceasta se face prin:

```
DO 2 I = 1,9
2 READ(10) (A(I, J), J = 1,100)
```

Tot la organizarea secvențială mai menționăm faptul că pe utilizator nu îl interesează lungimea blocurilor ci doar lungimea totală a fișierului. Lungimea totală îl ajută să aprecieze cît ocupă pe disc fișierul său și deci cam cît spațiu să aloce. În acest spațiu trebuie să intre și metainformația de informare pe care o folosește sistemul de operare. Sistemul de operare se ocupă și de lungimea blocurilor, aranjîndu-le în așa fel încît ele să nu depășească 256 baiți.

Din exemplul de mai sus apare problema introducerii unor modificări în organizarea fișierelor care să permită citirea directă de pe disc a înregistrării dorite, fără citirea celor anterioare. Organizarea care permite acest lucru este organizarea directă.

La organizarea directă sau, altfel spus, la fișierele în acces direct, instrucțiunea de scriere pe disc este *instrucțiunea WRITE cu acces direct*. Ea are următoarea formă:

```
WRITE (a'r, b) lista
```

Parametrii *a*, *b* și lista au semnificațiile de la instrucțiunile de citire/scriere secvențiale. Parametrul *r* este o variabilă întreagă, fără semn, care indică poziția înregistrării în setul de date (care formează un fișier) cu numărul de referință *a*. De exemplu, pentru înregistrarea numărul 9 din fișier, $r = 9$. Prin indicarea lui *a* (după tabelul 18), sistemul de operare știe adresa de început a fișierului (cilindrul și pista). Cunoscînd și pe *r*, poate fi citită în mod direct înregistrarea dorită. La depunerea fără format, *b* lipsește.

Instrucțiunea de citire de pe disc la organizarea directă este *READ (a'r, b) lista*, unde *a*, *b* și lista au aceeași semnificație. Ca și la organizarea secvențială, la organizarea directă este obligatoriu ca, dacă depunerea s-a făcut cu format, citirea să se facă cu același format. De asemenea, este obligatoriu ca lista de variabile de la citire să fie identică cu cea de la scriere. Aceasta înseamnă că la o citire de pe disc se citește dintr-odată blocul întreg.

Scanned with OKEN Scanner

Fie fișierul generat de $A(20, 100)$ organizat direct, cu variabilele grupate pe linii. Depunerea și citirea se face fără format. Atunci fiecare element al matricei A ocupă patru baiți și deci lungimea unei linii (pe disc un bloc) este $100 \times 4 = 400$ baiți. Tabelul 17 arată că încap cîte șapte blocuri pe o pistă. Pentru cele 20 de blocuri ale fișierului sînt necesare aproape trei piste. Dacă numele simbolic este *SYS007*, atunci $a = 10$ și scrierea pe disc a lui A se face prin

```
    DO 1 I = 1,20
1   WRITE (10'I)(A(I, J), J = 1, 100)
```

Pentru că ne referim la setul de date cu numărul de referință 10, era mai sugestiv dacă foloseam în loc de I variabila $I10$, ea fiind un contor pentru înregistrările fizice ale fișierului.

Citirea înregistrării numărului 9 se face prin instrucțiunile :

```
    I10 = 9
    READ(10'I10) (A(I10, J), J = 1,100)
```

La organizarea directă, utilizatorul *trebuie să cunoască* și *să indice* prin programul său lungimea blocurilor pe care le realizează prin grupare. Declararea lungimii blocurilor se va vedea la instrucțiunea *DEFINE FILE* și la operația de pregătire a discului pentru crearea fișierului organizat direct. În încheiere, mai menționăm că variabila contor r este mărită automat de sistemul de operare după ce s-a prelucrat (citire sau scriere) un bloc al fișierului.

TABELUL 18

Nr. de referință *a*	Nume simbolic	Numele fișierului
1	*SYSIN*	*IJSYSIN*
2	*SYSPCH*	*IJSYSPH*
3	*SYSLST*	*IJSYSLS*
4	*SYS001*	*IJSYS01*
5	*SYS002*	*IJSYS02*
6	*SYS003*	*IJSYS03*
7	*SYS004*	*IJSYS04*
8	*SYS005*	*IJSYS05*
9	*SYS006*	*IJSYS06*
10	*SYS007*	*IJSYS07*
11	*SYS008*	*IJSYS08*
12	*SYS009*	*IJSYS09*
13	*SYS010*	*IJSYS10*
14	*SYS011*	*IJSYS11*
15	*SYSLOG*	—

4.2.2. Instrucțiunile de definire și de descriere a fișierelor pe discuri

Ele sînt instrucțiuni de Job Control. Pentru că sînt date în general prin cartele, ne vom referi la ele prin denumirea de *cartele de comandă pentru fișiere*. Cartelele de co-

Scanned with OKEN Scanner

mandă pentru fișierele pe discuri descriu fișierele, ajută la adresarea, identificarea și protejarea lor. Ele au forme specifice versiunii sistemului de operare folosit, dar în așa fel încît versiunea nouă acceptă și formele specifice ale versiunii anterioare.

Cartelele de comandă pentru fișierele pe discuri sînt *VOL*, *DLAB*, *XTENT* și încep din coloana 1 (cu excepția cartelei *DLAB* care se continuă pe cartela următoare începînd din coloana 16). Orice fișier poate avea două nume: un nume pentru unitatea centrală și un nume pentru utilizator.

Cartela *VOL* are forma:

// *VOL SYS*xxx numele fișierului

*SYS*xxx este un nume simbolic al utilizatorului, asignat prin cartela de asignare unei unități de discuri. Numele fișierului este pentru unitatea centrală. În FORTRAN acest nume rezultă din tabelul 18.

Cartela *DLAB* are forma:

// *DLAB' numele fișierului*	1 *seria'*,	C
XXXX, AAZZZ, AAZZZ,'	*cod'*	

Cartela începe din coloana 1. Primul apostrof este în coloana 9. Numele fișierului este un nume pentru utilizator, în care acesta își descrie conținutul fișierului. De exemplu, *MATRICEA DISTANTELOR*.

Cifra 1 se afla în coloana 54 și indică folosirea codului *EBCDIC* în codificarea informației. Seria indică prin șase caractere alfanumerice sau numerice seria pachetului de discuri. Dacă pachetul are o altă serie, diferită de cea din *DLAB*, programul este respins de către unitatea centrală. În coloana 61 se pune apostrof, iar în 62 se pune virgulă. Litera *C* din coloana 72 indică continuarea informației pe cartela următoare, începînd cu coloana 16. *XXXX* indică numărul de ordine al volumului pe care se găsește fișierul. *AAZZZ* indică data zilei cînd a fost creat fișierul, sub forma anul-ziua. Exemplu: 71135. Următorul grup *AAZZZ* indică data expirării fișierului. Apostroful care urmează este în coloana 33. Codul este codul sistemului de operare. El este format din 13 caractere, după care urmează un apostrof în coloana 47. La *DOS*, codul poate rămîne blank. După aceste informații mai poate apare un parametru care indică tipul organizării și al prelucrării fișierului. Dacă acest parametru există, el este precedat de o virgulă în coloana 48.

Scanned with OKEN Scanner

Tipul se indică prin:

SD — organizare și prelucrare secvențială;

DA — organizare directă și prelucrare în acces direct;

ISC — organizarea indexat-secvențială, creare de fișier;

ISE — organizare indexat-secvențială, extindere de fișier. Dacă tipul este omis, se subînțelege *SD*.

Cartela *XTENT* definește mărimea și locul fișierului pe disc. Ea are forma:

// XTENT 1, secvența, lim inf, lim sup, 'seria', SYSxxx

Secvența indică numărul de ordine al *XTENT*-ului într-un fișier multiextent. Ea ocupă 1—3 coloane și este formată din numere de la 0 la 255.

Lim inf indică limita inferioară de la care începe fișierul (inclusiv cilindrul și pista indicată în limită). Ea este formată din nouă cifre și anume *OOOCCCOOP* unde *CCC* reprezintă cilindrul de început al fișierului (000—199), iar *P* indică pista de început a fișierului (0—9). Această formă a limitei este specifică unităților 2311.

Lim sup indică limita superioară pînă la care se întinde fișierul (inclusiv cilindrul și pista indicată). Ca formă, ea este identică cu limita inferioară.

Seria dintre apostroafe indică seria pachetului de discuri.

SYSxxx este numele simbolic căruia i s-a asignat o unitate de discuri și este ultimul cîmp de pe cartela *XTENT*.

Menționăm că, pentru programele scrise în FORTRAN, sistemul de operare cere ca pista de început a fișierului să fie pista zero a cilindrului.

Exemplu. Fie un fișier creat în ziua 150 din 1971, care expiră în ziua 200 din 1971 și care se întinde de la începutul cilindrului 5 la sfîrșitul cilindrului 8. Dacă pachetul de discuri este inițializat cu seria 654321, montat pe unitatea 193, care are numele simbolic *SYS001*, cartelele de comandă pentru fișierul numit *MATRICEA DISTANTELOR* sînt

```
// ASSGN SYS001, X' 193'
// VOL SYS001, IJSYS01
// DLAB' MATRICEA DISTANTELOR 1654321", C
                                0001, 71150, 71200,"
// XTENT 1,0,000005000,000008009, '654321', SYS001
```

Grupul de cartele *VOL*, *DLBAB*, *XTENT* este echivalent cu cartele *DLBL* și *EXTENT*. Descrierea lor se găsește în [3].

Scanned with OKEN Scanner

4.2.3. Instrucțiunea *DEFINE FILE* și pregătirea discului pentru crearea unui fișier în acces direct

Instrucțiunea *DEFINE FILE* este prima instrucțiune dintr-un program FORTRAN de prelucrare a unui fișier pe disc în acces direct (organizat direct). Ca orice instrucțiune FORTRAN pe cartelă, ea începe din coloana 7 și are următoarea formă:

*DEFINE FILE a*1 (*n*1, *l*1, *f*1, *I*1) *a*2(*n*2, *l*2, *f*2, *I*2)...

ai reprezintă *numărul de referință al setului de date.* El rezultă din tabelul 18, fiind numărul din stînga numelui simbolic al utilizatorului, *SYSxxx*, ales arbitrar. Numărul *ai* este identic cu *a* din *READ* și *WRITE* în acces direct, cînd acesta se referă la setul de date *ai*, adică la fișierul *ai*.

ni reprezintă *numărul de blocuri* asociate cu *ai*, adică numărul de blocuri din fișierul *ai*.

li reprezintă *lungimea unui bloc* și se măsoară în: caractere (baiți), locuri de memorie (baiți) și unități de memorie (cuvinte de patru baiți). *li* depinde de *fi*.

fi poate fi una din literele *L*, *E* sau *U*, care au semnificațiile:

L — citirea sau scrierea se face cu sau fără controlul formatului, iar lungimea blocurilor se măsoară în locuri de memorie;

E — indică citirea sau scrierea cu controlul formatului. Lungimea blocurilor se măsoară în caractere. Fiecare caracter de pe cartelă necesită un bait;

U — indică citirea sau scrierea fără controlul formatului. Lungimea blocurilor se măsoară în unități de memorie.

Ii este un contor de numărare a celor *ni* blocuri din fișierul referit prin numărul *ai*.

Exemple. Instrucțiunea *DEFINE FILE* 4(50, 80, *L*, *I*4) definește un fișier cu numele simbolic *SYS*001, referit prin numărul de referință 4. Fișierul conține 50 de blocuri, fiecare de cîte 80 baiți. Datele sînt prelucrate cu sau fără format, iar contorul de numărare al blocurilor este *I*4. Cei 80 de baiți pot să reprezinte, de exemplu, un bloc de 16 variabile reale grupate, fiecare variabilă ocupînd pe cartelă cîte cinci coloane (deci cinci baiți). Instrucțiunea format este deci *FORMAT* (16*F*5.2). Am presupus că variabilele au cîte două cifre după virgulă, pe cartelă fiind perforat și punctul zecimal. Dacă punctul zecimal nu este perforat pe cartelă, va trebui totuși să se țină cont de el în lungimea *li*, pentru că ocupă și el un bait. (Instrucțiunea format de mai sus putea să aibă și altă formă, de exemplu *FORMAT* (3I5, 13*F*5.1), dacă presupunem că grupăm trei variabile întregi și 13 variabile reale).

Scanned with OKEN Scanner

La prelucrarea în acces direct fără format (*U*), lungimea *li* este dată în unități de memorie. Numărul de unități de memorie necesar rezultă din

$$\left[\frac{\text{lungimea în baiți a blocului}}{4}\right],$$

unde prin [ ] am notat funcția partea întreagă. Dacă împărțirea se face exact, atunci *li* este chiar acest cît, iar dacă nu se face exact, *li* este partea întreagă a cîtului, mărită cu 1. De exemplu, fișierul definit de *DEFINE FILE* 7(50, 100, *L*, 17) devine *DEFINE FILE* 7(50, 25, *U*, *I*7), iar fișierul definit de *DEFINE FILE* 8(60, 110, *L*, *I*8) devine *DEFINE FILE* 8 (60, 28, *U*, *I*8), pentru că 110/4 implică $27 + 1 = 28$ unități de memorie.

A rămas de prezentat problema pregătirii discului pentru crearea unui fișier în acces direct. Vom presupune că discul pe care îl primește utilizatorul de la centrul de calcul a fost inițializat. Pregătirea discului constă în curățarea (anularea conținutului) porțiunii indicate prin *XTENT*. În afară de aceasta, se anunță unității de comandă a discului care este lungimea blocurilor fișierului ce va fi creat. Curățarea discului se face prin activarea din *CIL* a programului de curățire, catalogat sub numele *CLRDSK*.

Cartelele de comandă pentru curățirea discului formează un program separat de programul de prelucrare. De exemplu, fie de pregătit un disc, inițializat cu 999999, montat pe unitatea 193. Fișierul va ocupa cinci cilindri, începînd cu cilindrul 21. Dacă numele simbolic este *SYS*001, cartelele de comandă ale programului de pregătire sînt următoarele (fiecare începe din coloana 1):

```
// JOB PREGATIREA DISCULUI 193 SYS001 999999
// ASSGN SYS001, X'193'
// VOL SYS001,UOUT
// DLAB 'PRETURI DE COST  1999999', C
                    0001,71140,71200,'
// XTENT 1,0,000021000,000025009,'999999',SYS001
// EXEC CLRDSK
// UCL B = (K = s, D = n), X'40', OY
// END
/ &
```


Scanned with OKEN Scanner

$K = s$ indică lungimea în baiți a cheii înregistrărilor fizice. Dacă înregistrările nu au cheie (cum am presupus pînă acum) atunci $K = 0$. $D = n$ indică lungimea în baiți a înregistrărilor fizice. Lungimea n din cartela *UCL* trebuie să fie echivalentă cu lungimea *li* indicată în program, în instrucțiunea *DEFINE FILE*, în sensul că n este chiar *li* dacă *fi* este *L* sau *E* și $n = li \times 4$ dacă *fi* este *U*. *X'40'* indică faptul că spațiul liber (blank) este caracter de umplere, adică discul în porțiunea indicată de *XTENT* se șterge.

După ce discul a fost pregătit, pe el se pot depune datele care formează fișierul și se pot prelucra aceste date. Dacă utilizatorul are nevoie de mai multe fișiere pentru un program de prelucrare, în cadrul aceluiași program de pregătire a discului se includ *pentru fiecare fișier* cartelele *ASSGN, VOL, DLAB, XTENT, EXEC, UCL, END,* iar la sfîrșitul întregului pachet de cartele se pune cartela /&. Fiecare fișier trebuie să fie ulterior descris prin *DEFINE FILE*. Această instrucțiune suplinește indicativul *DA* din *DLAB*. Programul FORTRAN de prelucrare a fișierelor, scris de utilizator, nu poate fi folosit decît după ce discul a fost pregătit.

În paragraful următor se dau două exemple de programe care prelucrează fișiere: unul secvențial și altul în acces direct. Din aceste exemple se va vedea și forma standard a pachetului de cartele pentru un program FORTRAN.

4.2.4. Exemple de fișiere pe disc, prelucrate prin programe scrise în *FORTRAN*

Vom da două asemenea exemple: unul prelucrat secvențial, iar celălalt în acces direct. Programele propuse sînt foarte simple și de aceea nu se indică schemele logice ale programelor, ci se dă doar o descriere a succesiunii pașilor problemelor.

Exemplul 1 *de prelucrare a unor date în acces secvențial.* Fie o matrice *A* cu cinci linii și patru coloane: *A*(5, 4). Elementele matricei sînt numere întregi cu cel mult trei cifre fiecare. Ele sînt perforate pe cartele, cîte o coloană pe fiecare cartelă. Să se tipărească la imprimantă matricea *A* și transpusa ei.

Desigur, există mai multe modalități de rezolvare a acestei probleme. Noi vom folosi următoarea rezolvare, în care pentru folosirea discului vom grupa elementele pe coloane.

1. Se citește de pe cartelă cîte o coloană, care apoi se depune grupat și fără format pe disc.
2. Se tipărește linie cu linie matricea *A* la imprimantă.

Scanned with OKEN Scanner

3. În locul rezervat în memoria internă primei coloane a lui A, se citește de pe disc fiecare bloc al fișierului și se tipărește la imprimantă.

Vom folosi ca nume simbolic al utilizatorului pe *SYS007*, căruia îi corespunde 10 ca număr de referință al setului de date. Elementele matricei A sînt următoarele:

74	−42	430	−5
328	351	31	40
5	063	−731	17
−31	−8	54	180
999	20	93	−451

Problema se rezolvă prin următorul program:

```
// JOB E E 1 DISC SECVENTIAL IBM 360/30 C.C.U.B.
// PAUSE VA ROG MONTATI UN DISC LA ADRESA 193
// OPTION LINK
// ASSGN SYS007, X'193'
// VOL SYS007, IJSYS07
// DLAB 'FISIER MATRICEA A                    1111111',
        0001, 71270, 71300,'
// XTENT 1, 0, 000010000, 000011009, '111111', SYS007
// EXEC FFORTRAN
      DIMENSION A(5,4)
      INTEGER A
C CITESTE COLOANELE MATRICEI A DE PE CARTELE
      DO 1 J = 1, 4
      READ(1,2) (A(I,J) I = 1, 5)
    2 FORMAT (5I4)
C SCRIE PE DISC GRUPAT FIECARE COLOANA
    1 WRITE (10) (A(I,J), I = 1, 5)
      END FILE 10
C SCRIE LA IMPRIMANTA LINIILE LUI A
      WRITE (3, 4)
    4 FORMAT (' ', I10, 'MATRICEA A'/)
      DO 3 I = 1, 5
      WRITE (3, 5) (A(I, J), J = 1, 4)
    5 FORMAT ('0', 4(2X, I4))
    3 CONTINUE
```


Scanned with OKEN Scanner

```
C CITEȘTE DE PE DISC IN COLOANA 1 ȘI TIPARESTE
      WRITE (3, 6)
    6 FORMAT ('0', T15, 'MATRICEA TRANSPUSA')
      DO 7 J = 1, 4
      READ (10) (A(I, 1), I = 1, 5 )
    7 WRITE (3, 8) (A(I, 1)I = 1, 5)
    8 FORMAT ('0', T10, 5(2X, I4))
      STOP
      END
/*
// EXEC LNKEDT
// EXEC        .
007403280005 — 0310999
— 04203510063 — 0080020
04300031 — 73100540093
—005004000170180 — 415
/*
/$
```

Rezultatul prelucrării acestui program și timpul cît a durat execuția sînt următoarele:

```
      MATRICEA A
          74     —42     430      —5
         328     351      31      40
           5      63    —731      17
         —31      —8      54     180
         999      20      93    —415
      MATRICEA TRANSPUSĂ
          74     328       5     --31     999
         —42     351      63      —8      20
         430      31    —731      54      93
          —5      40      17     180    —415
EOJ E X 1   09.46.00, DURATION 00.01.36
```

Exemplul 2 de prelucrare a unor date în acces direct. Vom lua ca model problema transportului, care în forma ei cea mai simplă necesită trei matrice: matricea A a distanțelor, matricea B a cantităților disponibile (oferte) și matricea C a cantităților cerute (cererei). Deci avem de prelucrat trei matrice: $A(m,n)$, $B(m)$, $C(n)$.

Scanned with OKEN Scanner

Presupunem că, în tipul de problemă studiat, dimensiunile maxime sînt $A(10, 50)$, $B(10)$, $C(50)$. În particular, vom rezolva problema propusă mai jos pentru $m = 4$, $n = 5$. Elementele matricelor sînt următoarele:

	A					B
	37,48	1,171	0,12	17,20	3,45	3840,7
	40	41,4	0,01	32,5	0,719	130,54
	43,19	15	14,7	12	5,2	300
	6,8	5,43	21	3,291	18	1120,8
C	3617	140	21	200	1280	

Pentru că matricea C are numai elemente întregi o vom numi IC. Numele fişierelor pentru utilizator sînt: *MATRICEA A*, *MATRICEA B*, *MATRICEA IC*. Numele simbolice ale utilizatorului sînt *SYS001*, *SYS002*, *SYS003*, asociate respectiv lui A, B şi C. Toate fişierele se află pe acelaşi disc, montat de exemplu la adresa 193. Datele vor fi perforate pe cartele fără virgula zecimală, cîte o linie pe fiecare cartelă pentru matricea A, o cartelă pentru matricea B şi o cartelă pentru matricea IC. (Virgula zecimală este suplinită prin program, prin instrucţiunea *FORMAT*. Atragem atenţia că virgula zecimală din scrierea noastră obişnuită trebuie convertită pentru maşină în punct zecimal, adică 7,31 implică 7.31). Vom scrie elementele fiecărei matrice cu acelaşi număr de elemente, determinat de numărul maxim de cifre care se găseşte înainte şi după virgula zecimală. În acest fel, elementele lui A se perforează în cartele pe cinci coloane, elementele lui B pe şase coloane, iar ale lui IC pe patru coloane. Pentru m şi n vom folosi M şi N.

Schema de descriere a problemei este următoarea:

a) Pregătirea discului.
b) Programul de creare şi prelucrare a fişierelor:
1. Se citesc M şi N de pe cartelă.
2. Se citesc A, B, IC de pe cartele.
3. Se tipăresc la imprimantă A, B, IC.
4. Se depun pe disc A, B, IC.
5. Se şterg A, B, IC din memoria principală.
6. Se tipăresc A, B, IC la imprimantă.
7. Se citesc de pe disc în memorie A, B, IC.
8. Se tipăresc la imprimantă datele citite de pe disc.

Scanned with OKEN Scanner

Fişierele folosesc înregistrări blocate, fără format: patru blocuri pentru A şi cîte un bloc pentru B şi IC. În acest caz, lungimile blocurilor pentru A, B şi IC sînt, respectiv, 20, 16 şi 20, pentru că fiecărei variabile din lista de intrare/ieşire i se alocă patru baiţi.

Programele pentru paşii a şi b sînt următoarele:

```
// JOB EX2 PREGATIRE DISC ACCES DIRECT IBM 360/30 C.C.U.B.
// PAUSE VA ROG MONTATI UN DISC LA ADRESA 193
// ASSGN SYS001, X'193'
// VOL SYS001, UOUT
// DLAB 'MATRICEA A                              1111111', C
          0001,71280,71300,'        '
// XTENT 1, 0, 000010000, 000011009, '111111' ,SYS001
// EXEC CLRDSK
// UCL B = (K = 0, D = 20) ,X'40', OY
// END
// ASSGN SYS002, 'X'193'
// VOL SYS002, UOUT
// DLAB 'MATRICEA B                              1111111',  C
          0001, 71280, 71300, '      '
// XTENT 1, 0, 000012000, 000013009, '111111', SYS002
// EXEC CLRDSK
// UCL B = (K = 0, D = 16), X' 40', OY
// END
// ASGN SYS003, X' 193'
// VOL SYS003, UOUT
// DLAB 'MATRICEA IA                             1111111',  C
          0001, 71280, 71300, '        '
// XTENT 1, 0, 000014000, 000015009, '111111', 'SYS003
// EXEC CLRDSK
// UCL B = (K = 0, D = 20), X'40', OY
// END
/$

// JOB E X 2 DISC ACCES DIRECT IBM 360/30 C.C.U.B.
// OPTION LINK
// EXEC FFORTRAN
    DEFINE FILE 4(4, 20, U, I4), 5(1, 16, U, I5), 6(1, 20, U,I6)
    DIMENSION A(10, 50), B(10), IC(50)
    READ (1,100) M, N
```


Scanned with OKEN Scanner

```
100 FORMAT (2I2)
C CITESTE DE PE CARTELE A, B, IC
      DO 1 I = 1, M
    1 READ (1, 101) (A(I, J), J = 1, N)
101 FORMAT (5F5.3)
      READ (1, 102) (B(I), I = 1, M)
102 FORMAT (4F6.2)
      READ (1, 103) (IC(J), J = 1, N)
103 FORMAT (5I4)
C SCRIE PE DISC A, B, IC
      I4 = 1
      DO 2 I = 1, M
    2 WRITE (4'14) (A(I, J), J = 1, N)
      I5 = 1
      WRITE (5'I5)  (B(I), I = 1, M)
      I6 = 1
      WRITE (6'I6) (IC(J), J = 1, N)
C TIPĂREȘTE LA IMPRIMANTA
      K = 1
    9 DO 3 I = 1, M
      WRITE (3, 104)(A(I, J), J = 1, N)
104 FORMAT ('0', 5(2X, F7.3))
    3 CONTINUE
      WRITE (3, 105)(B(I), I = 1, M)
105 FORMAT ('0', T10, 4(2X, F7.2))
      WRITE (3, 106) (IC(J), J = 1, N)
106 FORMAT ('0', T15, 5(2X, I4))
C ȘTERGE A, B, IC DIN MEMORIA PRINCIPALA
      GO TO (8,10), K
    8 DO 4 J = 1, N
      IC (J) = 0
      DO 4 I = 1, M
    4 A(I, J) = 0.0
      DO 5 I = 1, M
    5 B(I) = 0.0
C TIPĂREȘTE A, B, IC LA IMPRIMANTA
      DO 6 I = 1, M
    6 WRITE (3, 107) (A(I, J), J = 1, N)
107 FORMAT ('0', 5F4.1)
      WRITE (3, 108) (B(I), I = 1, M)
108 FORMAT ('0', T10, 4F4.1)
      WRITE (3,109) (IC(J), J = 1, N)
```


Scanned with OKEN Scanner

```
109 FORMAT ('0',T15,513)
C CITESTE DISCUL ȘI DEPUNE IN MEMORIA PRINCIPALA
      I4 = 1
      DO 7 I = 1, M
    7 READ (4'14) (A(I, J), J = 1, N)
      I5 = 1
      READ (5'15)(B(I), I = 1, M)
      16 = I
      READ (6' 16) (IC(J), J = 1, N)
C TIPARESTE LA IMPRIMANTA DATELE CITITE DE PE DISC
      K = K + 1
      GO TO 9
   10 STOP
      END
/*
// EXEC LNKEDT
// ASSGN SYS001, X'193'
// ASSGN SYS002, X'193'
// ASSGN SYS003, X'193'
// VOL SYS001, IJSYS01
// DLAB 'MATRICEA A                                    1111111',          C
                0001, 71280, 71300,'
// XTENT 1,0,000010000, 000011009, '111111', SYS001
// VOL SYS002, IJSYS02
// DLAB 'MATRICEA B                                    1111111',          C
                0001, 71280, 71300,'
// XTENT, 1,0,0000120(0,000013009, '111111',  SYS002
// VOI SYS002, IJSYS03
// DLAB 'MATRICEA IC                                   1111111',          C
                0001, 71280, 71300,'
// XTENT 1,0,000014000,000015009, '111111',  SYS003
// EXEC
0405
37480011710012017200034 50
40000414000001032500007 19
43190150001470012000052 00
08600054308000003291180 00
38407001305403000011208 0
36170140210002001280
/*
/$
```


Scanned with OKEN Scanner

Rezultatele prelucrării programului și durata executării lui sînt:

```
37.480      1.171      0.120      17.200      3.450
40.000     41.400      0.010      32.500      0.719
43.190     15.000     14.700      21.000      5.200
 8.600      5.430     80.000       3.291     18.000
         3840.70     130.54      300.00     1120.80
              3617     140        2100        200      1280
0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0
          0.0  0.0  0.0  0.0
                0    0  0    0    0
37.480      1.171      0.120      17.200      3.450
40.000     41.400      0.010      32.500      0.719
43.190     15.000     14.700      12.000      5.200
 8.600      5.430     80.000       3.291     18.000
         3840.70     130.54      300.00     1120.80
         3617        140        2100        200      1280
```

EOJ EX2 10.10.48, DURATION 00.01.36

4.3. CREAREA ȘI PRELUCRAREA FIȘIERELOR DE PE BENZI MAGNETICE

4.3.1. Inregistrarea informațiilor pe bandă. Instrucțiuni de manevrare a benzii

Banda magnetică este un dispozitiv de memorie externă. Depunerea și accesul la informația de pe bandă se face în mod secvențial. Informația efectivă a utilizatorului este încadrată în cîmpuri de informație (etichete), întocmite de sistemul de operare. Pentru întocmirea etichetelor, sistemul de operare folosește informația de descriere a fișierului, pe care utilizatorul o furnizează în instrucțiunea *TPLAB* sau *TLBL*.

Un fișier pe bandă conține:

a) eticheta purtătorului de informație, care ocupă 80 baiți și conține șirul de caractere *VOL*, urmat de seria benzii.

b) etichete ale informațiilor:

Scanned with OKEN Scanner

1. *HDR* care indică începutul fişierului;
2. *EOV* sau *EOF* care indică sfîrşitul volumului (End Of Volume), respectiv sfîrşitul fişierului (End Of File);
3. *UHL* (User Header Label) care indică începutul informaţiei propriu-zise a utilizatorului;
4. *UTL* (User Trailer Label) care indică sfîrşitul informaţiei propriu-zise a utilizatorului.

Eticheta a) ajută la indentificarea rolei de bandă, iar b) ajută la identificarea fişierului pe bandă. Sistemul de operare mai înscrie pe bandă o marcă a benzii, notată *TM* (Tape Mark). *TM* se găseşte după *UHL*, după *UTL* şi înainte de *EOF* sau *EOV*. La sfîrşitul unei role de bandă, sistemul de operare înscrie de două ori *TM*. Poziţia etichetelor de tipul a) şi b) se vede în figura 16.

Înregistrările logice care se înregistrează pe bandă pot să fie de lungime fixă, grupate sau negrupate, de lungime variabilă, grupate sau negrupate şi de lungime nedefinită. Lungimea unei înregistrări fizice pe bandă este suma lungimilor tuturor variabilelor care apar în lista de variabile a unei instrucţiuni de depunere sau de citire de pe bandă. Este obligatoriu ca depunerea şi citirea să se facă cu acelaşi format.

În continuare ne vom referi numai la prelucrarea fişierelor cu ajutorul programelor scrise în FORTRAN. Instrucţiunile de intrare/ieşire (depunere-citire) sînt respectiv *WRITE* — *READ* secvenţial. Depunerea şi citirea se pot face:

1. *fără format: WRITE* (a) sau *READ* (a), unde *a* este numărul de referinţă al setului de date care este prelucrat. El rezultă din tabelul 18 şi anume este numărul care se găseşte în stînga numelui simbolic al utilizatorului (nume ales arbitrar din coloana din mijloc).

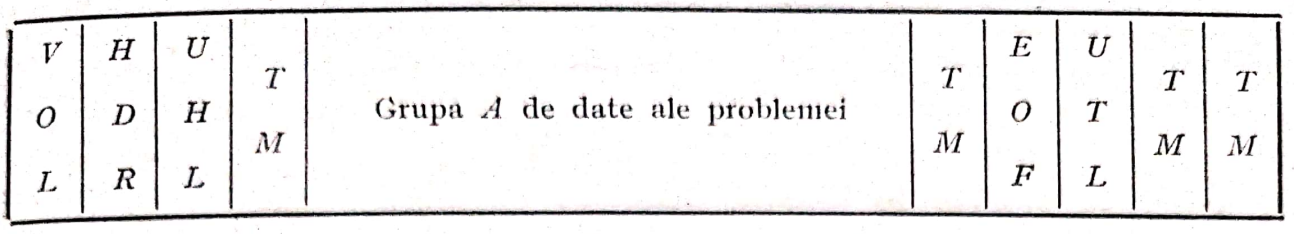
VOL	*HDR*	*UHL*	*TM*	Date ale problemei	*TM*	*EOF*	*UTL*	*TM*

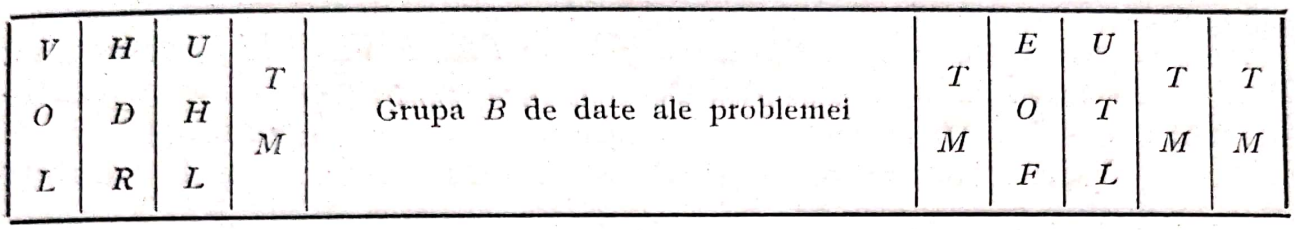
a) Fişierul ocupă o parte a rolei de bandă

VOL	*HDR*	*UHL*	*TM*	Date ale problemei	*TM*	*EOF*	*UTL*	*TM*	*TM*

b) Fişierul ocupă o rolă de bandă

Scanned with OKEN Scanner





c) Fişerul ocupă două role de bandă.

Fig. 16.

2. *cu format* : *WRITE* (*a*, *b*) sau *READ* (*a*, *b*) unde *a* are semnificaţia de mai sus, iar *b* este marca sau eticheta instrucţiunii *FORMAT*.

Gruparea înregistrărilor logice în blocuri se face prin lista variabilelor din instrucţiunile de intrare/ieşire, întocmai ca la prelucrarea informaţiei de pe discuri. Dacă depunerea informaţiei pe bandă se face fără format, fiecare variabilă din listă ocupă patru baiţi (dacă nu se specifică altfel în program).

În timpul prelucrării informaţiei de pe bandă, rola de bandă poate să fie manevrată conform dorinţelor utilizatorului. Aceste dorinţe pot indica :

— sfîrşitul fişierului, la crearea acestuia ;

— rebobinarea benzii, pînă la începutul fişierului, astfel încît dispozitivul de citire-scriere să se găsească la prima înregistrare logică din fişier ;

— întoarcerea înapoi a benzii (rebobinare) cu un bloc. Aceste operaţii sînt indicate de utilizator prin programul său, cu ajutorul următoarelor instrucţiuni :

1. instrucţiunea *END FILE*. Ea are forma

END FILE a

unde *a* reprezintă numărul de referinţă al setului de date prelucrat. Acest parametru este identic cu parametrul din instrucţiunea *WRITE* de creare a fişierului. Această instrucţiune indică sfîrşitul fişierului *a*.

2. instrucţiunea *REWIND* care determină rebobinarea benzii pînă la începutul fişierului *a*. Forma ei este

REWIND a

Scanned with OKEN Scanner

3. instrucțiunea *BACKSPACE* care determină întoarcerea înapoi a benzii cu un bloc al fișierului *a*. Forma ei este

BACKSPACE a

Pentru ca aceste instrucțiuni să fie executate de calculator, trebuie ca ele să fie precedate de una sau mai multe instrucțiuni *READ* sau *WRITE*. Astfel, instrucțiunea *END FILE* este executată dacă ea este precedată de o instrucțiune *WRITE*. *END FILE* determină înscrierea lui *TM* la sfîrșitul informațiilor propriu-zise ale utilizatorului. Această instrucțiune este ignorată dacă ea este precedată de *READ*. Instrucțiunile *REWIND* și *BACKSPACE* sînt ignorate dacă nu sînt precedate de *READ*, *WRITE* sau *END FILE*. Dacă *REWIND* sau *BACKSPACE* sînt precedate de *WRITE*, înaintea rebobinării (complete sau cu o înregistrare), se înscrie pe bandă *TM*.

Din punctul de vedere al utilizatorului, lungimea unei înregistrări fizice (bloc) din fișier este oricît de mare o permite rola de bandă sau partea din rolă rămasă utilizabilă, la un moment dat. Din punctul de vedere al sistemului de operare, lungimea unui bloc nu poate depăși 260 de baiți. De aceea, blocul informației utilizatorului este împărțit *automat* în *segmente de bloc* de cel mult 260 baiți. Într-un cîmp de doi biți rezervat fiecărui segment de bloc, sistemul de operare înscrie unul din următoarele coduri, cu următoarea semnificație:

00 — segmentul este primul și ultimul din bloc;

01 — segmentul este primul dar nu și ultimul din bloc;

10 — segmentul este ultimul dar nu și primul din bloc;

11 — segmentul nu este nici primul nici ultimul din bloc.

Deci codurile indicate arată legătura unui segment de bloc cu alte segmente din același bloc. Utilizatorul nu trebuie să se preocupe de împărțirea unui bloc cu mai mult de 260 de baiți în segmente de bloc, acest lucru fiind făcut automat de sistemul de operare.

4.3.2. Instrucțiunile de definire și de descriere a fișierelor de pe benzi

Aceste instrucțiuni sînt prelucrate de programul Job Control pentru întocmirea etichetelor. Ele nu sînt necesare la prelucrarea fișierelor în FORTRAN. Totuși le vom descrie pentru a putea fi folosite cu alte limbaje de programare.

Banda poate accepta etichete numai după ce ea a fost inițializată. Inițializarea se face cu un program utilitar de inițializare. Această operație o face operatorul sistemului de calcul.

Scanned with OKEN Scanner

În funcție de sistemul de operare *DOS* folosit pentru definirea și descrierea fișierului, se folosesc instrucțiunile *VOL* și *TPLAB* sau numai instrucțiunea *TLBL*. Sistemul de operare care acceptă instrucțiunea *TLBL* acceptă și grupul *VOL, TPLAB* dar nu și invers. În continuare ne vom referi la instrucțiunile date pe cartele.

Cartela *VOL* este identică cu cea descrisă la fișierele pe discuri.

Cartela *TPLAB* începe din coloana 1 și are forma:

// TPLAB 'nume

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

↑ 28 ↑ 34 ↑ 38 ↑ 42 | ↑ 46 ↑ 48 ↑ 54

nume	indică prin 17 caractere numele fișierului și este un identificator de fișier;
XXXXXX	indică prin șase caractere seria volumului;
XXXX	indică numărul de ordine al volumului în fișier;
XXXX	indică numărul de ordine al fișierului în volum;
XXXX	indică numărul generației fișierului;
XX	indică versiunea generației fișierului;
XXXXXX	indică data creerii fișierului; primul caracter rămîne gol pentru că data are forma *b AAZZZ*;
XXXXXX	indică data expirării fișierului.

Exemplu:

// TPLAB' FISIER PRETURI 11111100010001000101 71200 71350'

Am presupus că banda a fost inițializată cu 111111.

Cartela *TLBL* are forma:

// TLBL nume, ['nume ∪'], [data], [seria], [nr. volum], [nr. fișier], [generația], [versiunea]

nume	este un cîmp de 1—7 caractere și indică numele fișierului pentru sistemul de calcul; este luat din tabelul 2;
nume ∪	este numele fișierului pentru utilizator și ocupă 17 caractere între apostrofuri. Dacă acest operand este omis, el este înlocuit cu nume;
data	este data expirării fișierului. Se exprimă cu 4—6 caractere sub forma *ZZZZ* sau *AA/ZZZ*. Dacă acest operand este omis, fișierul expiră după o zi. Data creerii fișierului este data zilei curente;

Scanned with OKEN Scanner

seria — specifică prin șase caractere seria volumului. Dacă lipsește, sistemul de operare nu verifică seria rolei de bandă montată ;

nr. volum — ocupă patru caractere și indică numărul de ordine al volumului în fișier ;

nr. fișier — este analog cu operandul anterior ;

generația — ocupă patru caractere și indică generația fișierului. Dacă oricare din ultimii trei operanzi lipsesc, operandul se completează cu 0001 ;

versiunea — indică prin două caractere versiunea generației fișierului. Dacă lipsește, se completează cu 01.

Cartelele de comandă pentru fișierele pe benzi se pun în pachetul de cartele după cartela // *EXEC LNKEDT*. Înainte de această cartelă se pune cartela // *LBLTYP TAPE*, care rezervă o cantitate de memorie principală pentru prelucrarea etichetelor. Această cartelă nu este necesară dacă nu apar instrucțiuni de descriere și definire a fișierelor.

În paragraful următor se dă un exemplu de prelucrare printr-un program FORTRAN a unor date folosind banda.

În finalul acestui paragraf, menționăm faptul foarte important ca instrucțiunile *END FILE*, *REWIND* și *BACKSPACE* descrise mai sus sînt valabile și necesare și la prelucrarea sevcențială pe disc. Acest fapt explică apariția instrucțiunii *END FILE* 10 în programul de prelucrare secvențială pe disc.

4.3.3. Exemplu de folosire a benzii magnetice într-un program scris în FORTRAN

Ne propunem să creem pe bandă un fișier de cel mult 500 de numere, fiecare număr fiind pătratul unui număr natural. Numerele ale căror pătrate se calculează le presupunem consecutive și deci se formează unul din celălalt prin program (altfel se citesc de pe cartele). Reținem pătratele calculate în matricea *IPATR* (500) din memoria principală. Apoi formăm un fișier grupat pe bandă. Depunerea pe bandă și citirea de pe bandă se fac fără format. Descrierea programului este următoarea :

1. Se calculează pătratele a N numere naturale (N este citit de pe o cartelă, $N \leqslant 500$) și se reține în matricea *IPATR*.
2. Se tipăresc rezultatele la imprimantă.
3. Se crează pe banda montată pe unitatea 180 fișierul cu numele simbolic asociat. *SYS*003.
4. Se anulează în memorie conținutul lui *IPATR*.

Scanned with OKEN Scanner

5. Se citesc de pe bandă în *IPATR* datele fişierului creat.
6. Se tipăreşte la imprimantă conţinutul lui *IPATR*.

Programul care realizează succesiunea de operaţii de mai sus, pentru calcularea pătratelor numerelor 100, 101, ..., 109, este următorul:

```
// JOB EX3 BANDA MAGNETICA IBM 360/30 C.C.U.B.
// OPTION LINK
// PAUSE VA ROG MONTATI O BANDA PE UNITATEA 180
// ASSGN SYS003, X'180'
// EXEC FFORTRAN
      DIMENSION IPATR (500)
      READ (1, 10) N
   10 FORMAT (13)
      J = 100 + N - 1
      DO 1 I = 100, J
      IPATR (I-99) = I*I
      WRITE (3, 11) IPATR (I-99)
    1 CONTINUE
C CREAZA FISIERUL PE BANDA SYS003
      WRITE (6) (IPATR(1-99), I = 100, J)
      END FILE 6
C ANULEAZA IN MEMORIE CONŢINUTUL LUI IPATR
      DO 2 I = 100, J
    2 IPATR(I - 99) = 0
C SCRIE LA IMPRIMANTA CONTINUTUL LUI IPATR
      DO 3 I = 100, J
    3 WRITE (3, 11) IPATR (I-99)
   11 FORMAT (' ', I 6)
C CITEŞTE IN MEMORIE CONŢINUTUL FIŞIERULUI
      READ (6) (IPATR (I-99), I = 100, J)
C SCRIE LA IMPRIMANTA CANTITĂŢILE CITITE
      DO 4 I = 100, J
    4 WRITE (4, 11) IPATR (I-99)
      STOP
      END
      /*
      // EXEC LNKEDT
      // EXEC
      010
      /*
      /$
```


Scanned with OKEN Scanner

Rezultatul prelucrării acestui program și timpul cît a durat execuția sînt următoarele:

```
10000
10201
10404
10609
10816
11025
11236
11449
11664
11881
EOJ EX3        09.41.13, DURATION 00.01.55
```

BIBLIOGRAFIE

[1] IBM System/360 *DOS — Fortran IV, Programmer's Guide*, C28—6397—0
[2] IBM System/360 *DOS — System Control and System Service Programs*, C24—5036—3
[3] IBM System/360 *DOS — Operating Guide*, C24—5022—3
[4] IBM System/360 — *Principles of Operation*, A22—6821—6.
[5] IBM System/360 — *Fortran IV Language*, C28—6515—7
[6] IBM System/360 — *Direct Acces Storage Devices and Organization Methods* — Student Text.

Scanned with OKEN Scanner

Capitolul V

PROBLEME REZOLVATE

5.1. APLICAREA PROGRAMULUI LPC LA REZOLVAREA UNEI PROBLEME DE PROGRAMARE LINIARĂ

Să se rezolve următoarea problemă de programare liniară:

$$
\begin{aligned}
&2x_1 - x_2 + x_3 \leqslant 12,\\
&x_1 + x_4 \leqslant 5,\\
&2x_1 + 2x_2 + x_3 - 2x_5 \leqslant 20, \qquad (1)\\
&x_1 - x_2 - 2x_3 + 2x_4 - 2x_5 \leqslant 10,\\
&-2x_1 + 2x_2 - 2x_3 - 2x_4 + x_5 \leqslant 24,\\
&x_i \geqslant 0,\ i = 1, \ldots, 5, \qquad (2)\\
&\max\ (-2x_1 - 3x_2 + 2x_4 + 3x_5). \qquad (3)
\end{aligned}
$$

Pentru rezolvarea acestei probleme vom folosi programul de bibliotecă *LPC* (Linear Programming Code) [3]. Cu acest program se pot rezolva probleme liniare, teoretic, cu oricît de multe variabile. În ce privește α, numărul de restricții, acesta depinde de capacitatea memoriei și anume $\alpha \leqslant 230$ pentru o memorie de cel mult $32K$ și $\alpha \leqslant 4096$ pentru o memorie mai mare. Pentru aplicarea programului *LPC* este necesar să se cunoască:

a) modul de construire a fișierelor programului;
b) parametrii programului;
c) modul de introducere a datelor problemei;
d) interpretarea rezultatelor pe care le livrează calculatorul.

În cele ce urmează descriem fiecare din punctele de mai sus.

a) Există cinci fișiere asociate numelor simbolice *SYS009—SYS013*, create în exemplul dat pe un pachet de discuri montat pe unitatea 192. Mărimile fișierelor sînt date de cartelele *EXTENT* și sînt respectiv de 70, 70, 700, 70, 70 piste și încep, respectiv, la pistele 100, 200, 300, 1100, 1200. Fișierele construite în acest mod pot fi utilizate și pentru alte probleme.

b) Parametrii programului *LPC* pe care trebuie să-i cunoască orice utilizator sînt: *XOBJ*, care indică numele funcției obiectiv, *XRNS*

Scanned with OKEN Scanner

care indică numele coloanei termenilor liberi, *XDATA* care indică numele setului de date de intrare care vor fi prelucrate, *XPBNAME* prin care utilizatorul desemnează numele pe care îl dă problemei de rezolvat. Toţi aceşti parametri au cel mult opt caractere, sînt la dispoziţia utilizatorului şi se indică aşa cum se vede în exemplul rezolvat. Valorile asignate parametrilor *XDISKMT*, *XDISKSC*, *XDISKPB* şi *XDISKET* indică cilindrul de început, respectiv al fişierului *WORK MATRIX*, *SCRATCH*, *PROBFILE* şi *ETA*. Parametrul *XSCALE* = = −1. indică rezolvarea unei probleme de maxim, iar *XSCALE* = +1. indică rezolvarea unei probleme de minim. După cartela *XEND*, se pune cartela *NAME* care conţine numele dat prin *XDATA*, perforat din coloana 15.

În secţiunea de program *ROWS* se indică numele dat fiecărei restricţii a problemei şi caracterul lor. Numele se perforează din coloana 5, cel mult pînă în coloana 12. Caracterul restricţiilor se indică prin următorii indicatori: *E* înseamnă =, *L* înseamnă $\leqslant$, *G* înseamnă $\geqslant$, *UP* înseamnă limita superioară, iar *LO* înseamnă limita inferioară. Pentru funcţia obiectiv se pune *N*, care indică că ea nu are restricţii. Secţiunea *ROWS* se scrie aşa cum se vede în exemplul dat.

c) Introducerea datelor problemei coincide cu secţiunile *COLUMNS* şi *RHS*. În secţiunea *COLUMNS* se dă cîte un nume fiecărei coloane a matricei problemei de rezolvarat. Apoi se indică numai valorile nenule de la intersecţia coloanelor cu fiecare linie, începînd cu funcţia obiectiv. Numele coloanei are cel mult opt caractere şi se perforează din coloana 5. Din coloana 15, pe aceeaşi cartelă se indică numele funcţiei obiectiv sau numele liniei cu care coloana are intersecţie nenulă. Între 25 şi 36 se indică valoarea de la intersecţie, adică coeficientul problemei. Orice număr se perforează ca o valoare reală, adică numărul întreg este urmat de punct. În spaţiul 25—36 se include şi semnul minus, care se perforează în coloana 25. Există însă şi o altă modalitate de a scrie coeficienţii şi anume punctul zecimal se perforează în coloana 29, dacă numărul de cifre dinainte de punct permite aceasta. Această scriere permite programului o mai rapidă convertire a datelor. Pe această cartelă, de la 40 la 47 se perforează numele următoarei linii cu care coloana are intersecţie nenulă, iar între 50 şi 61 se perforează valoarea de la intersecţie. Ca şi mai înainte, valoarea poate fi perforată (împreună cu semnul) începînd din 50 sau, dacă este posibil, punctul zecimal se perforează în coloana 54. Pe cartelele următoare se continuă indicarea intersecţiilor nenule. Se vede că pe fiecare cartelă se perforează doi coeficienţi ai problemei.

Secţiunea *RHS* indică întocmai ca şi secţiunea *COLUMNS* valorile de la intersecţia coloanei termenilor liberi cu liniile matricei coeficienţilor.

Scanned with OKEN Scanner

Pentru exemplul rezolvat am întocmit următorul tabel.

		c_1	c_2	c_3	c_4	c_5	*LIB*
N	*FQB*	−2.	−3.		2.	3.	
L	*L1*	2.	−1.	1.			12
L	*L2*	1.			1.		5.
L	L3	2.	2.	1.		−2.	20.
L	*L4*	1.	−1.	−2.	2.	−2.	10.
L	*L5*	−2.	2.	−2.	−2.	1.	24.

În continuare dăm programul complet de rezolvare al problemei propuse şi rezultatele obţinute de calculator.

d) *Interpretarea rezultatelor.* Rezultatele prelucrării programului sînt tipărite la imprimantă. Programul anunţă dacă problema are soluţie optimă sau nu. În cazul în care soluţia optimă există, secţiunea *SOLUTION (OPTIMAL)* indică valoarea funcţiei de optimizat obţinută pentru soluţia optimă: $FQB = 184.0000$. Secţiunea *SECTION 1 — ROWS* dă indicaţii asupra satisfacerii restricţiilor problemei pentru soluţia optimă şi anume coloana *ACTIVITY* indică valoarea realizată în membrul stîng al restricţiei. Prin *SLACK ACIVITY* se notează variabilele de egalizare din metoda simplex. Coloana *UPPER LIMIT* indică coloana termenilor liberi. Se vede că suma elementelor corespunzătoare de pe coloanele *ACTIVITY* şi *SLACK ACTIVITY* conduce tocmai la valorile înscrise în coloana *UPPER LIMIT*. În cazul problemei rezolvate se vede că restricţiile *L1*, *L2* şi *L5* sînt satisfăcute prin egalitate, iar celelalte sînt satisfăcute prin $<$. Secţiunea *SECTION 2 — COLUMNS* dă soluţia optimă a problemei. Prin *BS* şi *LL* se notează variabilele de bază, respectiv variabilele secundare şi deci soluţia optimă este: $x_1 = 0$, $x_2 = 0$, $x_3 = 12$, $x_4 = 5$, $x_5 = 58$. Coloana *INPUT COST* conţine coeficienţii funcţiei de optimizat. Coloana *REDUCED COST* conţine diferenţele $c_j - z_j$ din tabelul simplex al soluţiei optime. Pentru variabilele secundare, aceste diferenţe sînt strict negative pentru problema de maxim şi strict pozitive pentru problema de minim. Coloana *DUAL ACTIVITY*

Scanned with OKEN Scanner

FORMULAR FORTRAN

CENTRUL DE CALCUL AL UNIVERSITATII
IBM 360/30

Program		Codificare												Pag. 1
		Semnul	[	]	=	+	ϕ							
Programator	Data	Se va perfora					zero							

	Instrucțiuni	Marcaj
1 Nr 5	7 10 15 20 25 30 35 40 45 50 55 60 65 70 72	73 75 80
	// JOB LPC FOLOSIREA PROGRAMULUI DE BIBLIOTECA LPC IBM 360/30	
	// ASSGN SYSØØ9, X'192'	
	// DLBL IJSYSØ9, 'FICHIER LABEL'	
	// EXTENT SYSØØ9,111111,1,Ø,Ø1ØØ,ØØ7Ø	
	// ASSGN SYSØ1Ø, X'192'	
	// DLBL IJSYS1Ø, 'FICHIER WORK MATRIX'	
	// EXTENT SYSØ1Ø,111111,1,Ø,Ø2ØØ,ØØ7Ø	
	// ASSGN SYSØ11, X'192'	
	// DLBL IJSYS11, 'FICHIER SCRATCH'	
	// EXTENT SYSØ11,111111,1,Ø,Ø3ØØ,Ø7ØØ	
	// ASSGN SYSØ12, X'192'	
	// DLBL IJSYS12, 'FICHIER PROBFILE'	
	// EXTENT SYSØ12,111111,1,Ø,11ØØ,ØØ7Ø	
	// ASSGN SYSØ13, X'192'	
	// DLBL IJSYS13, 'FICHIER ETA'	
	// EXTENT SYSØ13,111111,1,Ø,12ØØ,ØØ7Ø	
	// EXEC PLINROOT	
	XOPTION=FILEPROT,CONVERT,PICTURE	
	XPUNCH=254Ø	
	XOBJ=FOB	

Scanned with OKEN Scanner

FORMULAR FORTRAN

CENTRUL DE CALCUL AL UNIVERSITAȚII
IBM 360/60

Program		Codificare											Pag 2
		Semnul	[	]	=	+	ø						
Programator	Data	Se va perfora					zero						

Nr	Instrucțiuni	Marcaj
	XRHS=LIB	
	XDATA=EXLPC	
	XPBNAME=NFPROB	
	XSYSMT=10	
	XSYSSC=11	
	XSYSPB=12	
	XSYSET=13	
	XDISKMT=020	
	XDISKSC=030	
	XDISKPB=110	
	XDISKET=120	
	XFREQINV=2	
	XPRICE=10	
	XSCALE=-1	
	XNUMREC=4	
	XTOLDJ=1.E-05	
	XTOLRHL=1.E-10	
	XTOLER=1.E-03	
	XTOLZR=1.E-19	
	XTOLCHK=1.E-06	

Scanned with OKEN Scanner

FORMULAR FORTRAN

CENTRUL DE CAICUL AL UNIVERSITAȚII
IBM 360/30

Program		Codificare												Pag. 3
		Semnul	[	]	=	+	ϕ							
Programator	Data	Seva perfora					zero							

i Nr 5	6	Instrucțiuni 7 10 15 20 25 30 35 40 45 50 55 60 65 70 72	Marcaj 73 75 80
XTOLE	T	A=1. E-ϕ5	
XDZPC	T	=1. E-ϕ5	
XTOLP	V	V=1. E-ϕ6	
XTOLV	=	1. E-ϕ6	
XEND			
NAME		EXLPC	
ROWS			
N	FQB		
L	L1		
L	L2		
L	L3		
L	L4		
L	L5		
COLUMNS			
C1		FQB -2. L1 2.	
C1		L2 1. L3 2.	
C1		L4 1. L5 -2.	
C2		FQB -3. L1 -1.	
C2		L3 2. L4 -1.	
C2		L5 2.	

FORMULAR FORTRAN

CENTRUL DE CALCUL AL UNIVERSITĂȚII
IBM 360/30

Program		Codificare											Pag 4
		Semnul	[	]	=	+	ϕ						
Programator	Data	Se va perfora					zero						

Comentariu	Continuare	Instrucțiuni	Marcaj
1 Nr 5	6	7 10 15 20 25 30 35 40 45 50 55 60 65 70 72	73 75 80
C3		L1 1. L3 1.	
C3		L4 -2. L5 -2.	
C4		FQB 2. L2 1.	
C4		L4 2. L5 -2.	
C5		FQB 3. L3 -2.	
C5		L4 -2. L5 1.	
RHS			
LIB		L1 12. L2 5.	
LIB		L3 2ϕ. L4 1ϕ.	
LIB		L5 24.	
ENDATA			
Ta			
I&			

Scanned with OKEN Scanner

SOLUTION (OPTIMAL)

TIME = MINS. ITERATION NUMBER = 3

... NAME ...	...ACTIVITY ...	DEFINED AS.
FUNCTIONAL	184.00000	FQB
RESTRAINTS		LIB

SECTION 1 — ROWS

NUMBER	... ROW ...	AT	...ACTIVITY...	SLACK ACTIVITY	.. LOWER LIMIT.	..UPPER LIMIT.	.DUAL ACTIVITY
1	FQB	BS	184.00000	184.00000—	NONE	NONE	1.00000
2	L1	UL	12.00000	.	NONE	12.00000	6.00000—
3	L2	UL	5.00000	.	NONE	5.00000	8.00000—
4	L3	BS	104.00000—	124.00000	NONE	20.00000	.
5	L4	BS	130.00000—	140.00000	NONE	10.00000	.
6	L5	UL	24.00000	.	NONE	24.00000	3.00000—

SECTION 2 — COLUMNS

NUMBER	.COLUMN.	AT	... ACTIVITY ...	..INPUT COST..	..LOWER LIMIT.	..UPPER LIMIT.	.REDUCED COST.
7	C1	LL	.	2.00000—	.	NONE	16.00000—
8	C2	LL	.	3.00000—	.	NONE	3.00000—
9	C3	BS	12.00000	.	.	NONE	.
10	C4	BS	5.00000	2.00000	.	NONE	.
11	C5	BS	58.00000	3.00000	.	NONE	.

Scanned with OKEN Scanner

din *SECTION* 1 — *ROWS* conține soluția problemei duale, cu semn schimbat. Pentru problema dată, duala este:

$$
\begin{aligned}
&(1') \qquad 2y_1 + y_2 + 2y_3 + y_4 - 2y_5 \geqslant -2,\\
&\qquad -y_1 + 2y_3 - y_4 + 2y_5 \geqslant -3,\\
&\qquad y_1 + y_3 - 2y_4 - 2y_5 \geqslant 0,\\
&\qquad y_2 + 2y_4 - 2y_5 \geqslant 2,\\
&\qquad -2y_3 - 2y_4 + y_5 \geqslant 3,\\
&(2') \qquad y_i \geqslant 0,\ i = 1, \ldots, 5,\\
&(3') \qquad \min(12y_1 + 5y_2 + 20y_3 + 10y_4 + 24y_5).
\end{aligned}
$$

Soluția optimă a acestei probleme este $y_1 = 6$, $y_2 = 8$, $y_3 = 0$, $y_4 = 0$, $y_5 = 3$.

Din acest exemplu se poate observa satisfacerea teoremei de echilibru sau teorema ecarturilor complementare, care spune că dacă o restricție din problema directă este satisfăcută prin egalitate, atunci variabila duală corespunzătoare este nenulă și reciproc. Indicații detaliate asupra folosirii programului *LPC* se găsesc în [3], [4], [5].

5.2. MODELE DE SIMULARE CU CALCULATORUL

5.2.1. Definiția simulării, modele de simulare

Termenul „Simulare" s-a impus în mai multe sensuri în știința și tehnica contemporană (vezi [1], [7]). Aici ne vom referi exclusiv la simularea cu calculatorul digital. NAYLOR ([6]) dă următoarea definiție simulării digitale. „*Simularea* este o tehnică numerică pentru realizarea unor experiențe cu calculatorul digital, experiențe pe anumite tipuri de modele matematice și logice, care descriu comportarea unei instituții sau a unui sistem economic (sau a unor componente ale acestora) pe perioade mari de timp".

Deși această definiție pare a fi restrînsă la probleme economice, metodologia simulării este aceeași și pentru problemele care se întîlnesc în alte științe sau activități sociale.

Există două variante importante ale simulării, anume *simularea de tip joc* (jocuri operaționale) și *analiza Monte-Carlo*.

Scanned with OKEN Scanner

Simularea de tip joc presupune existența unui conflict între doi sau mai mulți parteneri ; în anumite cazuri, unul din parteneri este *natura*, iar altul este *omul* care, datorită observării experimentelor de simulare, poate să testeze anumite ipoteze sau să dea decizii asupra sistemului în cauză. Luarea deciziilor cu privire la evoluția sistemului este puternic legată de precizia cu care s-a realizat descrierea logică a sistemului.

Analiza Monte-Carlo (vezi HAMMERSLEY și HANDSCOMB, [2]) este o tehnică de simulare pentru probleme care au caracter probabilistic. Se pot distinge două tipuri de astfel de probleme. În primul rînd există probleme care presupun *simularea* prin diverse procedee a unor *procese stochastice* sau *variabile aleatoare*. În speță, este vorba de generarea cu calculatorul electronic a unor *selecții* asupra variabilelor aleatoare și vectorilor aleatori.

În al doilea rînd se disting probleme cu caracter complet deterministic ce nu pot fi rezolvate ușor (sau chiar deloc) prin metode analitice, dar pentru care este posibil să se obțină soluții aproximative prin simularea unei variabile stochastice ale cărei momente, densitate de repartiție sau funcție de repartiție sînt legate prin relații funcționale de soluția modelului determinist.

De regulă, multe probleme practice au legătură atît cu simularea de tip joc cît și cu analiza Monte-Carlo, deoarece majoritatea lor presupun existența unor elemente aleatoare ce trebuie simulate. Din această cauză problemele generării cu calculatorul a variabilelor aleatoare și proceselor stochastice joacă un rol important în teoria simulării.

Deși pentru anumite probleme soluțiile obținute prin simulare au un grad mare de aproximație, există multe motive de a prefera rezolvarea lor pe calea simulării. Aici vom menționa numai faptul că modelele de simulare pot avea un grad mai mare de complexitate și se pot referi la ipoteze mai generale, iar în cazul simulării sistemelor ele pot produce experiențe pentru sisteme care nu au fost încă niciodată realizate fizic, permițînd astfel să se realizeze *predicții* asupra sistemelor.

Modelele de simulare se aseamănă mult ca structură cu modelele matematice, așa cum au fost prezentate la § 2.1.1. Elementele constitutive ale unui model de simulare sînt : *componentele*, *variabilele* (de intrare sau ieșire), *parametrii* (de intrare sau ieșire), *relațiile funcționale* care pot fi și *identități*, *caracteristicile operative* (care sînt în general ipoteze matematice de lucru). Variabilele modelului pot fi și aleatoare (de fapt acesta este cazul cel mai interesant pentru simulare). Pentru a explicita modul cum evoluează în timp sistemul simulat, este nevoie să se utilizeze o variabilă specială numită „timp ceas" sau,

Scanned with OKEN Scanner

simplu, „ceas". Cu ajutorul acestui *ceas* se poate menține în timp ordinea corectă a apariției *evenimentelor* sistemului descris de variabile sau parametri. Ca variabile de ieșire, modelul de simulare precizează și *statisticile* ce trebuie produse în vederea estimării parametrilor de ieșire.

Toate aceste elemente se află în legături logice bine determinate de structura sistemului de simulat. Succesiunea și ordinea în care se schimbă elementele sistemului definesc un *algoritm*, sau *schema logică* a *modelului de simulare*. Această schemă logică trebuie să precizeze, în termeni de elementele modelului de simulare, modul în care se produc și dispar *evenimentele* din sistemul real care se simulează și modul în care se schimbă *stările* componentelor.

La fiecare iterare a schemei logice vor *„apare"* sau *„dispare" evenimente;* momentul atins de simulare este deci reprezentat de variabila ceas. Construcția schemei logice este puternic legată de modul cum se concepe variabila ceas. Există două tipuri de ceas al simulării și anume *ceas cu creștere constantă* și *ceas cu creștere variabilă.*

În cazul *ceasului cu creștere constantă,* schema logică se construiește avînd la bază ideea *simulării* unui interval constant de timp cu care se va mări ceasul; schema logică va descrie cum trebuie examinat sistemul pe parcursul acelui interval de timp pentru a determina ce evenimente *apar* sau dispar și ce stări vor avea componentele sistemului.

Modelele de simulare cu ceas cu creștere variabilă se bazează pe aceea că drept creștere a ceasului se ia intervalul de timp necesar apariției primului eveniment „viitor".

În figurile 17 (a) și 17 (b) este prezentat modul cum variază ceasul T pe parcursul simulării și cum apar evenimentele E_i în raport cu cele două tipuri de creștere a ceasului.

Există argumente atît în favorarea uneia cît și în favoarea celeilalte metode de creștere a ceasului. Se pare că metoda bazată pe ceas constant ușurează într-un anumit sens construirea *algoritmului* (schemei logice) de simulare. În acest caz se poate întîmpla însă ca să existe creșteri ale ceasului care să nu corespundă nici unei apariții de evenimente. De aceea, din punctul de vedere al timpului de calcul, metoda ceasului variabil pare a fi mai eficientă.

În această secțiune vor fi prezentate cîteva modele de simulare care fac uz de unul din procedeele de avansare a ceasului menționate.

După natura elementelor lor, modelele de simulare se pot clasifica în *modele statice* (în care nu intervine timpul în mod explicit), *modele dinamice* (în care timpul este o variabilă), *modele deterministe* (în care toate variabilele sînt nealeatoare) și *modele stochastice* (în care cel puțin o variabilă este aleatoare). Cele mai interesante modele de simu-

Scanned with OKEN Scanner

lare sînt modelele dinamice și stochastice, din care cauză o mare parte din problemele simulării sînt legate de generarea variabilelor aleatoare, vectorilor aleatori și proceselor stochastice.

În încheiere, se impune să mai precizăm etapele în care se rezolvă o problemă legată de cunoașterea (sau proiectarea) unui sistem real prin metoda simulării. Aceste etape sînt:

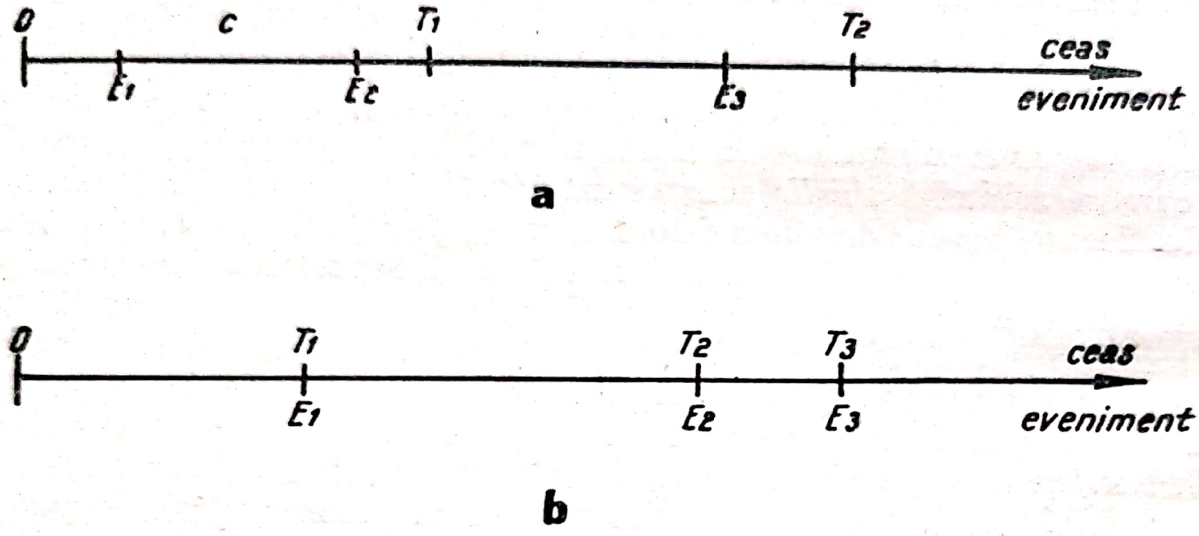


Fig. 17. a) Ceas cu creștere constantă; b) Ceas cu creștere variabilă.

(*i*) Formularea problemei.

(*ii*) Culegerea datelor și prelucrarea lor primară.

(*iii*) Formularea modelului de simulare al problemei.

(*iv*) Estimarea parametrilor caracteristicilor operative și testarea ipotezelor statistice.

(*v*) Analiza complexității modelului și a performanțelor lui.

(*vi*) Formularea unui program de calcul.

(*vii*) Validarea modelului de simulare.

(*viii*) Planificarea experiențelor de simulare.

(*ix*) Analiza datelor simulate.

Majoritatea etapelor menționate se autoexplică. Ordinea unora dintre ele este arbitrară.

Primele două etape pot fi eventual omise uneori dacă cu alte ocazii s-au formulat întrebările la care să răspundă simularea și s-au adunat observații suficiente. Vom face cîteva precizări numai asupra unora din ele.

Scanned with OKEN Scanner

(*iii*) Această etapă constă în a stabili componentele, variabilele, parametrii, relațiile funcționale, caracteristicile operative ale modelului și construirea schemei logice a acestuia.

(*iv*) În această etapă se estimează parametrii statistici de intrare ai modelului, corespunzător cu ipotezele făcute asupra variabilelor aleatoare de intrare ; se testează apoi aceste ipoteze statistice pe baza testelor corespunzătoare (teste de semnificație, teste de concordanță, teste neparametrice, etc.).

(*v*) Înainte de a trece la programarea modelului trebuie să se facă o *evaluare* a acestuia în sensul de a se constata dacă modelul este suficient de reprezentativ pentru sistemul de studiat sau dacă nu cumva numărul parametrilor și variabilelor este prea mare. Legat de aceasta, trebuie să se facă o evaluare și în ce privește performanțele modelului pe calculator (estimarea memoriei necesare a timpului de calcul, a volumului de *statistici* ce trebuie tipărite, etc.). Dacă în această etapă se constată că modelul nu este corespunzător dintr-un anumit punct de vedere, atunci se pot relua etapele (*i*) — (*v*) de rezolvare a problemei prin simulare.

(*vi*) Scrierea programului de simulare se poate face într-un limbaj general (de ex. FORTRAN) sau într-un limbaj de simulare. Există *limbaje de simulare* specifice diferitelor probleme. De exemplu, limbajul *GPSS* (prescurtare de la General Purpose Sistems Simulation) se folosește la simularea fenomenelor care presupun *așteptări* sau *aglomerări*, pe cînd limbajul *PROGRAM SIMULATE* este destinat simulării sistemelor economice.

Trebuie menționat că alegerea limbajului poate interveni chiar la construirea schemei logice (etapa (*iii*)). O schemă logică pentru simularea unui sistem va arăta într-un fel dacă modelul urmează să fie programat în FORTRAN și în alt fel dacă modelul urmează să fie programat în *GPSS*.

Trebuie însă în același timp precizat că programarea într-un limbaj general asigură programului o anumită *flexibilitate* care nu se realizează cu limbajele de simulare.

(*vii*) După ce programul a fost testat sintactic trebuie ca acesta să fie supus și unei testări *logice*. Mai precis, trebuie să constatăm dacă modelul rezolvă *bine problema* pentru care este destinat. Dacă, pentru întrebările la care trebuie să răspundă simularea, există răspunsuri analitice pentru forme particulare ale caracteristicilor operative, atunci *validarea* modelului se poate face *comparînd* în acele cazuri particulare rezultatele analitice cu cele simulate. Dacă însă astfel de soluții analitice nu există, atunci singura cale de validare este de a *compara* rezultatele simulate cu observațiile asupra sistemului real de simulat. Lucrurile se complică atunci cînd se simulează un

Scanned with OKEN Scanner

sistem care din punct de vedere fizic nu a mai fost realizat. Cum însă orice sistem fizic complicat este compus din *componente*, unele deja realizate, validarea trebuie să se realizeze măcar pentru acele componente ale sistemului.

(*viii*) Planificarea experienţelor de simulare constă în precizarea valorilor de intrare a *parametrilor de intrare*; valorile acestor parametri pot *varia* în diversele rulări ale unui program de simulare cu scopul de a se stabili valorile care realizează un anumit *optim* pentru unele variabile sau parametri de ieşire, sau un optim al unei funcţii criteriu, definită ad-hoc. Problema planificării experienţelor de simulare este de fapt *problema statistică a planificării experienţelor*. Există o bogată literatură legată de planificarea experienţelor statistice (vezi [8]).

(*ix*) Rezultatele simulării se prezintă de obicei sub formă de *selecţii statistice* asupra variabilelor de ieşire. Aceste date statistice trebuie analizate cu ajutorul metodelor statisticii matematice (teoria estimaţiei, verificarea ipotezelor, etc.). Trebuie însă avut în vedere faptul că rezultatele simulării se prezintă sub forma unor date corelate care se schimbă în timp; deci este necesar ca pentru interpretarea lor să se aplice metode ale statisticii proceselor stochastice.

În continuare vom prezenta unele modele concrete de simulare.

5.2.2. Model de simulare pentru studiul endemiei tuberculozei

Aşa cum s-a văzut şi în § 2.1.1, modelul matematic pentru studiul endemiei tuberculozei nu este satisfăcător, întrucît el poate fi rezolvat analitic numai în cazuri particulare, atunci cînd funcţiile (7) — (13) (§ 2.1.1) sînt liniare. Transformînd însă modelul matematic în *model de simulare directă*, se poate elimina ipoteza liniarităţii funcţiilor f_i, $1 \leqslant i \leqslant 7$. În modelul de simulare directă vom considera variabilele din membrul drept al relaţiilor (7) — (13) pe care le vom nota X_i, $1 \leqslant i \leqslant 7$, drept *variabile aleatoare*, ale căror funcţii de repartiţie se presupun cunoscute. Dacă există algoritmi pentru *generarea* unor valori de selecţie ale acestor variabile aleatoare, atunci valorile funcţiilor I, B, N la momentul *următor* se pot obţine din valorile lor la momentul *actual* utilizînd valorile de selecţie ale varabilelor X_i, $1 \leqslant i \leqslant 7$ în relaţiile (1) — (3) (§ 2.1.1). Rezumînd, algoritmul de calcul al modelului de simulare va fi:

0. Intrare: I_0, B_0, N_0 (valorile funcţiilor la momentul $t = 0$).
   Intrare: parametri necesari generării variabilelor X_i, $1 \leqslant i \leqslant 7$.
1. Generează X_i, $1 \leqslant i \leqslant 7$.

Scanned with OKEN Scanner

2. Calculează I_1, B_1, N_1 din relațiile de recurență.

$I_1 = (1 - \alpha)I_0 + X_1 + X_3 - X_5 - X_2$;

$B_1 = B_0 + X_2 - X_3 - X_6$;

$N_1 = N_0 + X_4 - X_1 - X_7 + \alpha I_0$.

3. $P_1 = I_1 + B_1 + N_1$.

La o iterare a pașilor 1—3 ai algoritmului precedent se obține cîte o valoare a variabilelor (aleatoare) I_1, B_1, N_1; iterînd de mai multe ori acești pași, se pot estima mediile și deviațiile standard ale variabilelor respective la momentul $t = 1$. Pentru a se trece apoi la momentul următor ($t = 2$, etc.) trebuie să se adauge pasul

4. $I_0 = I_1$, $N_0 = N_1$, $B_0 = B_1$, transfer la 1.

Dacă programul de calcul tipărește mediile și deviațiile standard ale variabilelor I_1, N_1, B_1, obținute prin repetarea pașilor 1—3 și apoi se execută 4, atunci se pot obține rezultate statistice referitoare la cele trei variabile, rezultate care reprezintă soluția modelului. Numărul de iterări ale pasului 4 corespunde numărului de ani pentru care se obțin valorile funcțiilor aleatoare de timp $I(t)$, $N(t)$, $B(t)$.

Rămîne însă de precizat, pentru modelul în discuție, modul de generare a variabilelor aleatoare X_i, $1 \leqslant i \leqslant 7$. Întrucît α ia valori într-un interval de forma (0, 1), iar X_i, $1 \leqslant i \leqslant 7$, iau valori întregi, ele pot avea printre altele una din următoarele tipuri de repartiții: uniformă, triunghiulară, Beta — PERT, normală sau log-normală (ultimele două — trunchiate) pentru α și Poisson pentru X_i. Fără a intra în detalii, vom prezenta în rezumat cîțiva algoritmi de generare a variabilelor aleatoare care au aceste repartiții. Se presupun cunoscute unele elemente de statistică matematică.

a) *Repartiția uniformă.* Variabila aleatoare X are repartiția uniformă discretă dacă ea ia valorile $a_1, \ldots, a_n$, fiecare cu probabilitatea $\frac{1}{n}$. Se pot genera cu calculatorul valori de selecție pentru o variabilă de forma

$$X : \begin{pmatrix} 0 & 1 & 2 & \ldots & M-1 \\ \frac{1}{M} & \frac{1}{M} & \frac{1}{M} & \ldots & \frac{1}{M} \end{pmatrix},$$

care este o variabilă uniformă întreagă cu M valori. De exemplu, dacă $M = 2^{31}$ și $a = 65539$, astfel de selecții se generează cu relația de recurență

(4) $$X_{i+1} \equiv aX_i \pmod{M}, \quad X_0 - \text{prim}.$$

Scanned with OKEN Scanner

Variabila aleatoare

$$U = \frac{X}{M} \tag{5}$$

este o variabilă uniformă pe (0, 1). Ea are densitatea de repartiție

$$f(u) = \begin{cases} 1, & \text{dacă } u \in (0, 1), \\ 0, & \text{în rest.} \end{cases}$$

Valori de selecție asupra lui U se obțin generînd mai întîi valori X și aplicînd apoi formula (5). Prin transformarea

$$V = a + (b - a)U \tag{6}$$

se obțin valori de selecție pentru variabila aleatoare uniformă pe un interval oarecare (a, b).

b) *Repartiția triunghiulară.* Se spune că variabila aleatoare X este triunghiulară [13] pe intervalul (a, b), avînd modul m dacă densitatea sa de repartiție este

$$f(x) = \begin{cases} \dfrac{2(x-a)}{(m-a)(b-a)}, & \text{dacă } a \leqslant x \leqslant m, \\ \dfrac{2(x-b)}{(b-m)(b-a)}, & \text{dacă } m \leqslant x \leqslant b \\ & \text{în rest.} \end{cases} \tag{7}$$

Funcția sa de repartiție este

$$F(x) = \begin{cases} 0, & \text{dacă } x \leqslant a, \\ \dfrac{(x-a)^2}{(m-a)(b-a)} & \text{dacă } a \leqslant x \leqslant m, \\ \dfrac{(x-b)^2}{(m-b)(b-a)} + 1, & \text{dacă } m \leqslant x \leqslant b, \\ 1, & \text{dacă } x \geqslant b. \end{cases} \tag{7'}$$

O metodă generală pentru generarea unei valori de selecție a variabilei aleatoare X care are funcția de repartiție $F(x)$ este *metoda inversă* care se bazează pe relația

$$X = F^{-1}(U),$$

unde F^{-1} este inversa lui F, iar U este o valoare de selecție a variabilei uniforme pe (0, 1) care se generează ca la punctul a). Deci, în cazul variabilei triunghiulare, generarea se face cu formula

Scanned with OKEN Scanner

$$(7'')\quad X = \begin{cases} a + \sqrt{(m-a)(b-a)\,U}, & \text{dacă } 0 \leqslant U \leqslant \dfrac{m-a}{b-a}, \\ b - \sqrt{(b-m)(b-a)\,U}, & \text{dacă } \dfrac{m-a}{b-a} \leqslant U \leqslant 1. \end{cases}$$

c) *Repartiția Beta-PERT*. Variabila aleatoare V, care are densitatea de repartiție

$$(8)\qquad f(x) = \begin{cases} \dfrac{1}{B(\nu_1,\ \nu_2)}\, x^{\nu_1 - 1}\,(1-x)^{\nu_2 - 1}, & x \in (0,\ 1), \\ 0, & \text{în rest,} \end{cases}$$

se numește *variabilă Beta* de parametri ν_1 și ν_2.
Variabila

$$(9)\qquad W = a + (b-a)V,\ W \in (a,\ b)$$

se numește *variabilă Beta-PERT*; generarea lui W se reduce deci la generarea lui V.

Se știe [14] că dacă $G(\lambda,\ \nu_1)$ și $G(\lambda,\ \nu_2)$ sînt două variabile aleatoare gama independente, atunci variabila

$$(10)\qquad V(\nu_1, \nu_2) = \frac{G(\lambda,\ \nu_1)}{G(\lambda,\ \nu_1) + G(\lambda,\ \nu_2)}$$

este o variabilă aleatoare *Beta* de parametri ν_1 și ν_2. Deci generarea lui V se reduce la generarea de variabile gama (deoarece λ nu joacă un rol esențial, se poate lua $\lambda = 1$).

Variabila aleatoare $G(\lambda,\ \nu_1)$ este o variabilă gama de parametri λ și ν dacă are densitatea de repartiție:

$$g(x) = \begin{cases} \dfrac{\lambda^{\nu}}{\Gamma(\nu)}\, x^{\nu-1}\, e^{-\lambda x}, & x \geqslant 0, \\ 0, & \text{în rest.} \end{cases}$$

Dacă ν este întreg atunci se arată cu ușurință că

$$(11)\qquad G(\lambda,\ \nu) = \sum_{i=1}^{\nu} E_i(\lambda),$$

unde $E_i(\lambda)$ sînt variabile aleatoare exponențiale de parametru λ, independente. Generarea variabilelor gama se reduce deci (cazul ν = natural) la generarea variabilelor exponențiale. (Pentru $\nu > 1$, oarecare, vezi [9]).

Scanned with OKEN Scanner

Variabila aleatoare $E(\lambda)$ este o variabilă exponențială de parametru λ dacă are densitatea de repartiție:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geqslant 0, \\ 0, & \text{în rest.} \end{cases} \tag{12}$$

Funcția de repartiție a lui $E(\lambda)$ este

$$P(E(\lambda) < z) = \int_0^z f(x)\mathrm{d}x = 1 - e^{-\lambda z};$$

deci, conform metodei inverse, generarea variabilei aleatoare $E(\lambda)$ se face cu formula

$$E(\lambda) = -\frac{1}{\lambda}\ln(1 - U). \tag{13}$$

Deoarece însă $1 - U$ are aceeași repartiție cu U, relația (10) se simplifică astfel:

$$E(\lambda) = -\frac{1}{\lambda}\ln U. \tag{13'}$$

Sînt astfel prezentate toate considerațiile referitoare la generarea variabilelor *Beta — PERT*.

d) *Repartiția normală*. Variabila aleatoare X are o repartiție normală $N(\mu, \sigma)$ dacă densitatea sa de repartiție este

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \qquad -\infty < x < \infty.$$

Dacă $U_1, \ldots, U_n$ sînt n variabile aleatoare independente, uniforme pe $(0, 1)$, atunci, conform cunoscutei teoreme limită centrală, variabila

$$Z_n = \frac{U_1 + \ldots + U_n - \frac{n}{2}}{\sqrt{\frac{n}{12}}}$$

are o repartiție normală $N(0, 1)$ cînd $n \to \infty$. În particular, pentru $n = 12$, se obține următoarea formulă simplă pentru generarea unei variabile $N(0, 1)$:

$$Z = U_1 + \ldots + U_{12} - 6. \tag{14}$$

Scanned with OKEN Scanner

Generarea unei variabile X normală $N(\mu, \sigma)$ se face deci cu formula

$$X = \mu + Z\sigma, \tag{15}$$

unde Z este definit de (14).

O variabilă normală X de parametri μ și σ, *trunchiată* pe (a, b), se generează prin următorul procedeu de *respingere:*

1. Se generează o variabilă y normală $N(\mu, \sigma)$.
2. Dacă $Y \notin (a, b)$, se *respinge* valoarea Y și se trece la pasul 1, altfel se trece la pasul 3.
3. Se ia $X = Y$.

e) *Repartiția lognormală.* Variabila aleatoare T are o repartiție lognormală de medie m și abatere standard s dacă $X = \ln T$ este o variabilă normală $N(\mu, \sigma)$. Se arată (vezi [13]) că

$$\mu = \ln m - \frac{1}{2} \ln \left[\left(\frac{s}{m} \right)^2 + 1 \right], \quad \sigma^2 = \ln \left[\left(\frac{s}{m} \right)^2 + 1 \right]. \tag{16}$$

Deci generarea variabilei T se face astfel:

0. Se calculează în prealabil μ și σ din (16).
1. Se generează o variabilă X normală $N(\mu, \sigma)$.
2. Se ia $T = e^X$.

Generarea variabilei lognormale, trunchiată pe intervalul (a, b), se face printr-un procedeu de respingere ca la pct. c).

Acum toate elementele modelului de simulare pentru studiul endemiei tuberculozei sînt precizate.

5.2.3. Model de simulare pentru problema interferenței mașinilor

Această problemă constă în următoarele. Se consideră N mașini care se defectează (se opresc) la intervale neregulate (aleatoare) de timp; un număr de M mecanici se ocupă de întreținerea lor ($M < N$), dar din cauză că uneori toți mecanicii sînt ocupați, unele din mașini vor trebui să *aștepte* eliberarea unui mecanic. Se cere să se găsească numărul M de mecanici care să asigure în modul cel mai *eficient* repararea celor N mașini, avîndu-se în vedere faptul că prin folosirea unui număr mare de mecanici s-ar putea întîmpla ca unii dintre ei să rămînă *neocupați* pe anumite perioade de timp.

Timpul de așteptare a mașinilor pînă la *eliberarea* mecanicilor care să le repare se numește timp de *interferență.*

În fapt, pentru problema în cauză se presupun cunoscute repartițiile timpilor de funcționare (repartiția venirilor) și timpilor de reparare (repartiția serviciului), se presupune că reparațiile se fac într-o

Scanned with OKEN Scanner

anumită ordine (de exemplu, în ordinea în care mașinile s-au defectat) și se caută să se obțină informații despre timpul de interferență și timpul de neocupare a mecanicilor.

Deci, problema la care ne referim este o problemă de *teoria cozilor*. Există modele matematice pentru multe probleme care presupun aglomerări. Pentru a distinge între aceste modele, Kendall (1951) și Lee (1966) au introdus notația

$$V/S/c:(L/d) \tag{17}$$

unde V se referă la veniri, S se referă la servicii, c reprezintă numărul de stații de serviciu (canale), L reprezintă lungimea maximă a cozii, iar d disciplina serviciului.

Dacă în (17) anumite elemente ale modelului sînt neprecizate, atunci se va înlocui poziția corespunzătoare cu un punct (de exemplu, dacă nu se știe repartiția serviciului se va pune punct în loc de V).

Problema interferenței mașinilor conduce deci la un model de așteptare de forma

$$././M:((N-M)/FI\ FO) \tag{18}$$

unde *FIFO* (prescurtare de la „first in first out") este disciplina de servire în ordinea opririi.

Modelul de simulare pentru problema interferenței mașinilor, definită de (18), este prezentat în lucrarea [10].

Variabilele și parametrii modelului, exprimate în notații FORTRAN, sînt

N = numărul de mașini;
M = numărul de mecanici;
I = lungimea cozii la un moment oarecare;
K = numărul de mecanici neocupați la un moment oarecare;
NB = numărul total de evenimente ce se vor simula (se face convenția că *venirile* și *serviciile* sînt evenimente);
RT = durata de funcționare (variabilă aleatoare) dintre două opriri consecutive;
ST = durata serviciului (variabilă aleatoare);
WT = timpul de așteptare al unei mașini;
TID = timpul de neocupare al tuturor mecanicilor;
$T(J)$ = timpul total al mașinii J (adică un fel de ceas al mașinii J, $1 \leqslant J \leqslant N$);
$SS(J)$ = timpul total solicitat pentru serviciu de mașina J, $1 \leqslant J \leqslant N$;
$SR(J)$ = timpul total de funcționare al mașinii J;

Scanned with OKEN Scanner

$SW(J)$ = timpul total de aşteptare al maşinii J;
$TAT(L)$ = timpul total al mecanicului L, $1 \leqslant L \leqslant M$ (un fel de ceas al mecanicului L);
$SID(L)$ = timpul total de neocupare al mecanicului L;
$TBS(T)$ = timpul total de lucru al mecanicului L;

$$IX(J) = \begin{cases} 0, \text{ dacă maşina } J \text{ lucrează,} \\ 1, \text{ dacă maşina } J \text{ este în reparaţie,} \\ -1, \text{ dacă maşina } J \text{ aşteaptă;} \end{cases}$$

JM = maşina de care se „ocupă" la un moment dat programul de simulare;
KM = maşina cu cel mai lung timp de aşteptare;
LM = mecanicul de care se „ocupă" la un moment dat programul de simulare;
LA = primul mecanic eliberat cînd toţi lucrează;
TMI = ceasul simulării (cu creştere variabilă);
$TATM$ = timpul total al mecanicului care se află în lucru (timpul mecanicului LM);
TMK = timpul total al maşinii cu cel mai lung timp de aşteptare;
$ICOUNT$ = un întreg care numără evenimentele.

Clasificarea variabilelor şi parametrilor este următoarea:

Întregii N, M, NB sînt parametri de intrare. ST şi RT sînt variabile aleatoare de intrare, cărora trebuie să le asociem caracteristici operative şi care vor trebui generate; parametrii lor vor fi de asemenea parametri de intrare. Variabilele $SS(J)$, $SR(J)$, $SW(J)$, $1 \leqslant J \leqslant N$, şi $SID(L)$, $TBS(L)$, $1 \leqslant L \leqslant M$, sînt variabile de ieşire. Celelalte variabile nu sînt esenţiale pentru modelul de simulare ci ele sînt variabile de lucru, introduse spre a facilita programarea modelului.

Variabilele modelului de simulare, introduse anterior, satisfac următoarele *condiţii iniţiale:*

$$\begin{aligned} &I = ICOUNT = 0,\ K = M, \\ &SS(J) = SW(J) = 0,\ 1 \leqslant J \leqslant N, \\ &TAT(L) = TBS(L) = SID(L) = 0,\ 1 \leqslant L \leqslant M, \\ &IX(J) = 0,\ 1 \leqslant J \leqslant N. \end{aligned} \tag{19}$$

Schema logică a modelului de simulare este prezentată în figura 18. Ea se bazează pe metoda ceasului cu creştere variabilă şi foloseşte notaţiile FORTRAN introduse anterior.

Blocul (1) citeşte parametrii de intrare ai modelului, iar blocul (2) precizează condiţiile iniţiale din formulele (19). În continuare, blocu-

Scanned with OKEN Scanner

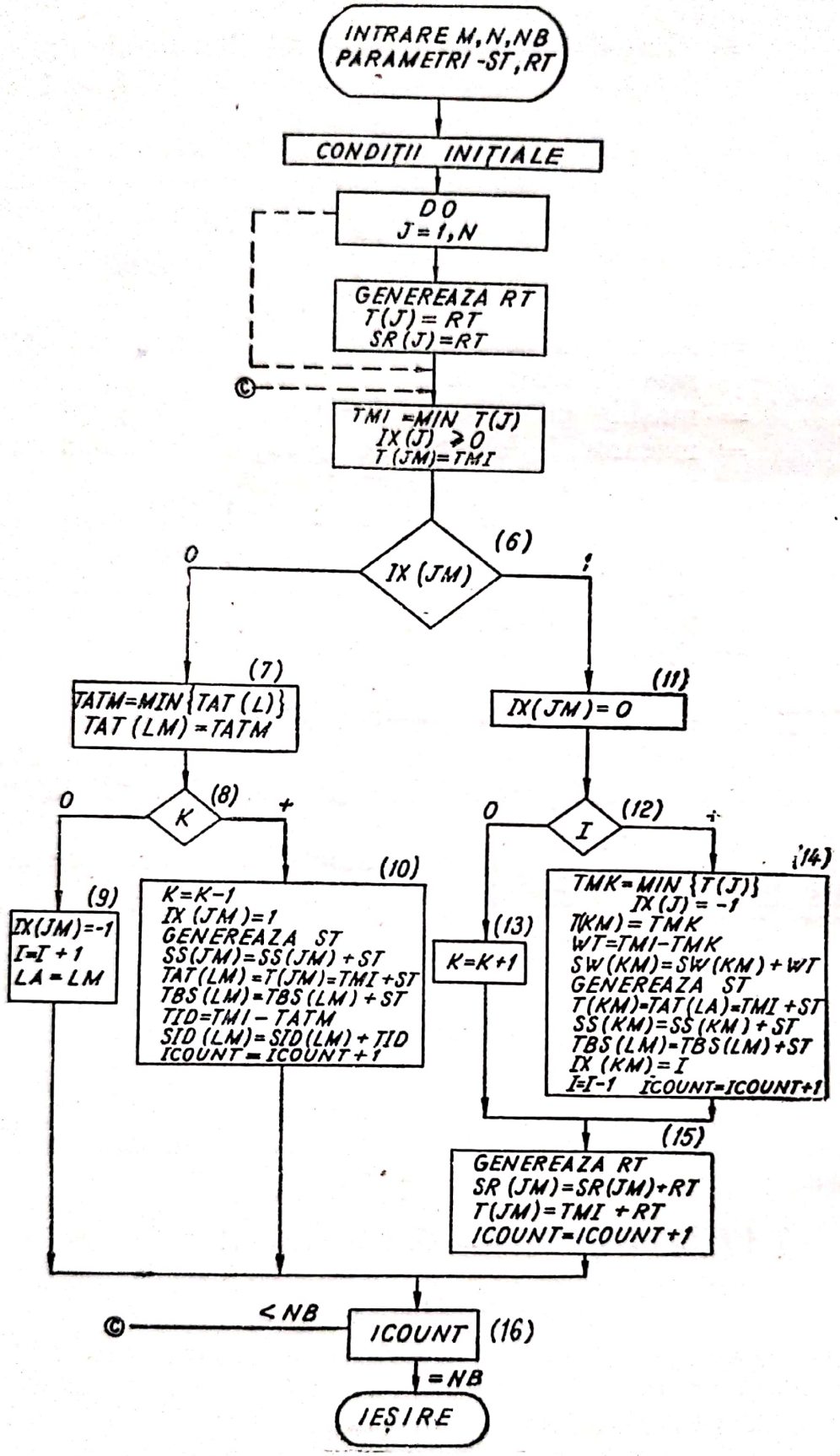


Fig. 18. Schema logică a modelului de simulare pentru problema interferenței mașinilor.

Scanned with OKEN Scanner

rile (3) și (4) simulează pasul inițial al procesului, cînd toate mașinile sînt pornite. Blocul (5) determină mașina care trebuie să fie luată în considerare la pasul următor al simulării avînd ca disciplină a serviciului *FIFO*; se folosește ceasul cu creștere variabilă. Astfel, mașina *JM* este mașina cu cel mai scurt timp *TMI* (simulat), dintre mașinile în lucru sau în curs de reparație; dacă *TMI* corespunde la două astfel de mașini, atunci mașina în curs de reparație se consideră prima. Blocul (6) testează dacă mașina *JM* este în funcțiune sau în curs de reparare. Dacă ea se află în lucru, atunci se execută blocurile (7) — (10); astfel, blocul (7) alege mecanicul (*LM*) care trebuie să fie ocupat la acel moment; apoi, blocul (8) testează dacă există un mecanic liber sau nu. În ultimul caz, blocul (9) *pune* mașina *JM* în așteptare, mărește lungimea cozii *I* cu o unitate, interpretează mecanicul *LM* ca primul care se eliberează și transferă apoi controlul la blocul (16). Dacă, dimpotrivă, în blocul (8) s-a găsit un mecanic liber, atunci se execută blocul (10); numărul mecanicilor liberi se micșorează cu o unitate și se *pune* mașina *JM* în reparație; timpul total al mașinii *JM*, timpul total de ocupare al mecanicului *LM* și timpul total al lui *LM* sînt crescute cu *ST* (durata reparației); se determină apoi timpul cît mecanicul *LM* a stat neocupat pînă la începerea reparației și se înregistrează acest timp la *SID*; se înregistrează apoi un nou pas al simulării prin mărirea lui *ICOUNT* cu o unitate. Revenind la blocul (6), dacă mașina *JM* este în reparație, atunci se execută blocurile (11) — (16). Blocul (11) pune mașina în funcțiune, eliberînd un mecanic. Dacă în blocul (12) se găsește că lungimea cozii este zero, atunci în blocul (13) numărul de mecanici liberi *K* crește cu o unitate și controlul trece la blocul (15). Dimpotrivă, dacă în blocul (12) coada nu este vidă, atunci se execută blocul (14): mai întîi, se alege mașina *KM* care a așteptat cel mai mult (conform disciplinei *FIFO*) și se determină timpul ei de așteptare *WT* care se adaugă la timpul său total de așteptare *SW*; în continuare, în blocul (15) se execută operații similare blocului (10), micșorîndu-se totodată lungimea cozii cu o unitate. Blocul (15) descrie starea mașinii *JM* (care fusese în reparație și acum este în funcțiune). Astfel, se generează un timp de funcționare *RT* care se înregistrează la timpul total al mașinii *JM* și apoi se înregistrează noul pas simulat.

În final, blocul (16) verifică dacă s-au simulat *NB* pași; în cazul cînd nu s-au realizat *NB* pași, se reiau operațiile începînd cu blocul (5).

Pentru validarea modelului de simulare se compară soluțiile simulate cu rezultatele analitice obținute într-un caz particular.

Se presupune că timpul *RT* este o variabilă aleatoare exponențială de parametru λ $\left(\text{valoarea medie a lui } T \text{ este } E(RT) = \frac{1}{\lambda}\right)$ și timpul

de serviciu ST are media $\frac{1}{\mu}$ $\left(\text{adică } E(ST) = \frac{1}{\mu}\right)$. Din teoria firelor de aşteptare (vezi [10]) se ştie că, dacă se notează

$$\rho = \frac{E(ST)}{E(RT)} = \frac{\lambda}{\mu}$$

intensitatea de trafic, atunci se poate calcula *factorul de interferenţă*

$$J = \frac{E(WT)}{E(ST)}. \tag{20}$$

Pentru ST, variabilă aleatoare constantă sau exponenţială (de parametru μ), cu $M = 1$ şi toate maşinile identice, se găseşte că

$$J = \frac{1}{y\rho} + N - 1 - \frac{1}{\rho}, \tag{21}$$

unde y are în fiecare caz o anume expresie. Cînd ST este constant (formula lui Ashcroft) se obţine

$$y_A = 1 + \sum_{K=1}^{N-1} \binom{N-1}{K} (e^{\rho} - 1)(e^{2\rho} - 1) \dots (e^{K\rho} - 1), \tag{22}$$

iar cînd ST este exponenţial (formula lui Palm) se găseşte

$$y_P = 1 + \sum_{K=1}^{N-1} (N-1)(N-2) \dots (N-k)\rho^K. \tag{22'}$$

Deci, din (21) şi (22), se obţine o valoare teoretică J_A, iar din (21) şi (22') se obţine o valoare teoretică J_P.

Cu ajutorul simulării se poate obţine şi o estimaţie a factorului de interferenţă J_E, după formula

$$I_E = \frac{\text{Timp total de aşteptare}}{\text{Timp total de serviciu}} = \frac{SWT}{SST}. \tag{23}$$

Rezultatele validării sînt prezentate în tabelul 19. S-au considerat drept parametri de intrare $\lambda = 1.0$, iar pentru μ s-au luat următoarele cazuri

1. $\mu = 5.0$ (serviciu potrivit),
2. $\mu = 10.0$ (serviciu rapid),
3. $\mu = 2.0$ (serviciu lent).

Scanned with OKEN Scanner

TABELUL 19

Modelul	Cazul	Nr. de paşi simulaţi NB	Factorul de interferenţă	
			Teoretic	Simulat
Ashcroft	1	8.000	1.035578	1.034116
	2	32.000	0.368253	0.349703
	3	8.000	3.004391	3.003210
Palm	1	5.000	1.424339	1.404621
	2	20.000	0.639522	0.660879
	3	5.000	3.073394	3.132766

Din analiza tabelului, rezultă că există o concordanţă satisfăcătoare între valorile teoretice şi cele rezultate din simulare în ce priveşte parametrul de ieşire J, deci modelul de simulare a fost construit în mod corect.

Modelul de simulare poate fi deci utilizat pentru rezolvarea problemei interferenţei maşinilor în cazul cînd $1 < M < N$, cu maşini neidentice (repartiţii ale timpilor de funcţionare diferite) şi mecanici neidentici (repartiţii ale timpilor de serviciu diferite). Se pot astfel calcula :

— factorul de eficienţă al maşinilor

$$EF = \frac{\sum_J SR(J)}{\sum_J (SR(J) + SW(J) + SID(J))}; \tag{24}$$

— factorul de eficienţă al serviciului (mecanicilor)

$$W = \frac{\sum_L TBS(L)}{\sum_L (TBS(L) + SID(L))}, \tag{25}$$

care în cazul general nu pot fi determinaţi analitic (prin formule).

5.3. CÎTEVA PROBLEME DE STATISTICĂ MATEMATICĂ

Aplicarea metodelor statisticii matematice la prelucrarea şi interpretarea datelor statistice necesită un volum mare de calcule. Din această cauză, mai ales în cazul multidimensional, o analiză statistică completă este posibilă numai pe baza utilizării calculatoarelor electronice.

Scanned with OKEN Scanner

Fiind dată o selecție $X_1, \ldots, X_n$, efectuată asupra unei variabile aleatoare sau vector aleator X, prelucrarea datelor statistice ridică de obicei probleme de felul următor:

— să se verifice o ipoteză statistică referitoare la repartiția teoretică a datelor statistice (ipoteza de concordanță);

— să se calculeze estimațiile parametrilor repartiției teoretice sau să se determine relațiile funcționale între acești parametri;

— să se verifice ipoteze asupra parametrilor statistici ai repartiției teoretice.

Vom prezenta aici două exemple de aplicații de acest fel care se întîlnesc în mod frecvent, dar a căror tratare nu apare în literatură,

5.3.1. Testul χ^2 pentru verificarea normalității multidimensionale

Fie X un vector aleator k dimensional. Presupunem că X are o repartiție normală $N(\mu, \Sigma)$, unde μ este un vector coloană $k \times 1$, iar Σ este o matrice pozitiv definită $k \times k$. Densitatea de repartiție a lui X este

$$f(x) = \frac{1}{\sqrt{(2\pi)^k}\sqrt{\det \Sigma}} e^{-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)}, \tag{26}$$

unde

$$\Phi(x) = (x-\mu)'\Sigma^{-1}(x-\mu)$$

este o formă pătratică pozitiv definită.

Fiind dată selecția $X_1, \ldots, X_n$ efectuată asupra lui X, se pune problema să se verifice ipoteza H că repartiția lui X este normală $N(\mu, \Sigma)$, cu μ și Σ necunoscuți.

Verificarea ipotezei H se face cu ajutorul testului χ^2 *de concordanță* care constă în următoarele (vezi [11], [14]):

— se alege o diviziune $\Delta_1 U \ldots U \Delta_l = \Delta$ a domeniului Δ al valorilor lui X (în cazul nostru Δ este spațiul R^k);

— fie $p_i(\alpha) = \text{Prob}(X \in \Delta_i)$, $1 \leqslant i \leqslant l$, unde prin α am notat parametrii μ și Σ;

— cu ajutorul selecției X_i, $1 \leqslant i \leqslant n$, se determină frecvența absolută ν_j pe domeniul Δ_j (adică numărul valorilor de selecție care cad în Δ_j);

— se estimează parametrii α din condiția

$$\sum_{i=1}^{l} \frac{(\nu_i - np_i(\alpha))^2}{np_i(\alpha)} = \min, \quad np_i(\alpha) = \text{const}, \quad 1 \leqslant i \leqslant l, \tag{27}$$

cunoscută sub numele de metoda minimului lui χ^2;

Scanned with OKEN Scanner

— notînd cu α^* estimaţia lui α, se calculează statistica

$$\chi^2 = \sum_{i=1}^{l} \frac{(\nu_i - n p_i^*)^2}{n p_i^*}, \quad p_i^* = p_i(\alpha^*), \tag{28}$$

care, dacă $n \to \infty$, are $f = l - c - 1$ grade de libertate, unde c este numărul parametrilor estimaţi, adică

$$c = \frac{k(k+3)}{2}.$$

Dacă, pentru pragul de semnificaţie π dat, se alege χ_0^2 astfel încît

$$P(\chi_f^2 < \chi_0^2) = 1 - \pi,$$

atunci ipoteza H se acceptă dacă $\chi^2 < \chi_0^2$ şi se respinge în caz contrar. Dacă l şi k sînt fixaţi astfel încît $l > c + 1$, valoarea χ_0^2 pentru π fixat se citeşte din tabelele funcţiei de repartiţie χ^2.

Pentru aplicarea testului χ^2 la verificarea ipotezei H sînt necesare cîteva precizări şi anume care sînt expresiile estimaţiilor lui μ şi Σ şi cum se calculează probabilităţile p_i ?

După cum se ştie (vezi [11]), estimaţiile lui μ şi Σ sînt date respectiv de formulele

$$\hat{\mu} = \bar{X} = \frac{1}{n_1} \sum_{i=1}^{n_1} X_i \tag{29}$$

$$\hat{\Sigma} = S = \frac{1}{n_1 - 1} A, \quad A = \sum_{i=1}^{n_1} (X_i - \bar{X})(X_i - \bar{X})',$$

unde n_1 este numărul observaţiilor folosite, $n_1 < n$. Dacă notăm

$$X_\lambda = (X_{1\lambda}, \ldots, X_{k\lambda})',$$

elementul a_{ij} al matricei A va fi

$$a_{ij} = \sum_{\lambda=1}^{n_1} (X_{i\lambda} - \bar{X}_i)(X_{j\lambda} - \bar{X}_j). \tag{29'}$$

Să notăm $N = n - n_1$ volumul selecţiei rămase după estimarea lui μ şi Σ. Presupunem că N este foarte mare şi $n_1 > k$.

Pentru a calcula p_i^*, folosim faptul că statistica

$$F_{k,n_1-k}^{(i)} = \frac{n_1(n_1 - k)}{(n_1 - 1)(n_1 + 1)k} (X_i - \bar{X})' S^{-1} (X_i - \bar{X}), \quad 1 \leqslant i \leqslant N, \tag{30}$$

este o variabilă aleatoare F cu k şi $n_1 - k$ grade de libertate, care se va nota cu $F_{k,\, n_1-k}$. Funcţia de repartiţie a acestei variabile este

$$\psi(\theta) = P(F_{k,\, n_1-k} < \theta) = \left(\frac{k}{n_1 - k}\right)^{\frac{k}{2}} \frac{\Gamma\left(\frac{n_1}{2}\right)}{\Gamma\left(\frac{k}{2}\right)\Gamma\left(\frac{n_1 - k}{2}\right)} \cdot$$

(31)

$$\cdot \int_0^{\theta} z^{\frac{k}{2}} \left(1 + \frac{k}{n_1 - k}\, z\right)^{-\frac{n_1}{2}} \delta z.$$

Notînd cu E_θ elipsoidul $F_{k,\, n_1-k} < \theta$ cu $F_{k,\, n_1-k}$ definit de (30), rezultă ca putem construi o diviziune a spaţiului $R^k = \Delta$, luînd $0 < \theta_1 < \theta_2 < \ldots < \theta_{l-1} < \theta_l = \infty$ şi $\Delta_1 = E_{\theta_1}$, $\Delta_i = E_{\theta_i} - E_{\theta_{i-1}}$, $2 \leqslant i \leqslant l$. Probabilităţile p_i^* sînt acum determinate cu ajutorul funcţiei $\psi(\theta)$, definită de (31).

Sîntem acum în măsură să precizăm algoritmul de calcul privind verificarea ipotezei H cu ajutorul testului χ^2. Algoritmul este următorul:

Pasul 0. Se construieşte mai întîi o subrutină pentru calculul funcţiei $\psi(\theta) = P(F_{f_1, f_2} < \theta)$, definită de relaţia (31).

Pasul 1. Intrare: k, n_1, π, l, χ_0^2; l se alege astfel încît $l > \frac{k(k+3)}{2} + 1$. Se alocă memoria pentru listele $\theta[1], \ldots, \theta[l-1]$; $P[1], \ldots, P[l]$ şi $\nu[1], \ldots, \nu[l]$. Se ia $\nu[i] = 0$, $1 \leqslant i \leqslant l$ şi se aleg $0 < \theta[1] < \ldots < \theta[l-1]$. Se ia $j = 0$.

Se alocă memoria pentru variabila logică H.

Pasul 2. Folosindu-se subrutina precizată la pasul 0, se calculează $P[1] = \psi(\theta_1)$, $P[i] = \psi(\theta_i) - \psi(\theta_{i-1})$, $2 \leqslant i \leqslant l-1$, $P[l] = 1 - \psi(\theta_{l-1})$.

Pasul 3. Se estimează μ şi Σ cu formulele (21), pe baza a n_1 valori de selecţie. $N = n_1 - k$.

Pasul 4. Citeşte o valoare de selecţie X, ia $j \leftarrow j + 1$.

Pasul 5. Calculează F conform formulei (30). Dacă $F < \theta[1]$, atunci $\nu[1] \leftarrow \nu[1] + 1$; dacă $\theta[i-1] \leqslant F < \theta[i]$, atunci $\nu[i] \leftarrow \nu[i] + 1$, $2 \leqslant i \leqslant l-1$; dacă $F \geqslant \theta[l-1]$, atunci $\nu[l] \leftarrow \nu[l] + 1$. (Acest pas determină frecvenţele).

Pasul 6. Dacă $j < N$, atunci se execută pasul 4.

Pasul 7. (Cînd $j = N$). Se calculează

$$\chi^2 = \sum_{i=1}^{l} \frac{(\nu[i] - NP[i])^2}{NP[i]}.$$

Scanned with OKEN Scanner

Pasul 8. Dacă $\chi^2 < \chi^2_0$, atunci H este adevărată, iar dacă $\chi^2 \geqslant \chi^2_0$, H este falsă.

Algoritmul pentru testul χ^2 este acum descris. În vederea aplicării lui, ar trebui construite subrutine pentru calculul estimaţiilor μ şi Σ (pasul 3), precum şi pentru calculul lui F (pasul 5). De fiecare dată este util să se ţină seama de faptul că matricele Σ (respectiv S) sînt simetrice; pentru calculul lui F este necesar să se determine în prealabil S^{-1}.

Exceptînd subrutina de calcul pentru ψ, toate celelalte subrutine necesare aplicării testului χ^2 descris mai sus se găsesc descrise în *SSP*.

5.3.2. Analiză dispersională bifactorială

Presupunem că o variabilă aleatoare normală unidimensională Y depinde de doi factori *calitativi* A şi B, avînd respectiv I nivele şi J nivele. Aceşti factori determină IJ *celule*. Presupunem că $\sigma^2 = \text{Var}(Y)$ este constantă pentru fiecare celulă, dar că fiecare celulă poate influenţa media variabilei aleatoare Y. Problemele analizei dispersionale bifactoriale constau în a estima mediile variabilei Y pentru cele IJ celule şi a verifica influenţele celulelor asupra mediilor, pe baza unor observaţii asupra lui Y. Vom nota cu Y_{ijk} cea de-a k — a observaţie asupra lui Y efectuată în celula (i, j) (cînd factorul A are nivelul i şi factorul B are nivelul j). Presupunem $1 \leqslant k \leqslant K$, adică în fiecare celulă se efectuează acelaşi număr K de experienţe. Acesta se mai numeşte şi cazul experienţelor echilibrate. Fie $\eta_{ij} = E(Y_{ijk})$ valoarea medie a lui Y în celula (i, j).

Deci (vezi ([11]) ne aflăm în următoarea ipoteză fundamentală

$$(32)\quad \Omega: \begin{cases} Y_{ijk} = \eta_{ij} + e_{ijk},\ 1 \leqslant i \leqslant I,\ 1 \leqslant j \leqslant J,\ 1 \leqslant k \leqslant K; \\ e_{ijk} \text{ sînt independente şi au repartiţii normale } N(0, \sigma); \\ \quad \sigma = \text{const.} \end{cases}$$

Variabilele aleatoare e_{ijk} (care reprezintă *erorile de observaţie*) sînt presupuse deci normale şi independente.

Pentru a studia influenţa factorilor asupra mediilor celulelor, presupunem că η_{ij} admite următoarea descompunere

$$(33)\quad \eta_{ij} = \mu + \alpha_i + \beta_j + \gamma_{ij},$$

unde μ reprezintă o medie absolută a variabilei Y, α_i reprezintă efectul nivelului i al factorului A asupra mediei lui Y, β_j reprezintă efectul nivelului j al factorului B asupra mediei lui Y, iar γ_{ij} reprezintă

Scanned with OKEN Scanner

efectul interacțiunii (sau pe scurt interacțiunea) nivelelor i și j asupra mediei lui Y. Se fac în plus ipotezele:

$$\sum_i \alpha_i = \sum_j \beta_j = \sum_i \gamma_{ij} = \sum_j \gamma_{ij} = 0 \tag{34}$$

care spun că efectele diverselor nivele ale factorilor se anulează reciproc.

Se cere deci ca, pe baza celor $N = IJK$ observații efectuate asupra lui Y, să se estimeze parametrii μ, α_i, β_j, γ_{ij} și să se verifice ipoteze de forma

$$\begin{aligned} &H_A: \alpha_i = 0, \ 1 \leqslant i \leqslant I; \quad H_B: \beta_j = 0, \quad 1 \leqslant j \leqslant J; \\ &H_{AB}: \gamma_{ij} = 0, \quad 1 \leqslant i \leqslant I, \quad 1 \leqslant j \leqslant J. \end{aligned} \tag{35}$$

Semnificația ipotezelor ce trebuie verificate este evidentă. Dacă, de exemplu, H_A este adevărată, atunci înseamnă că efectele nivelelor factorului A asupra mediei η sînt nule; cu alte cuvinte, clasificarea observațiilor asupra lui Y după factorul A este neesențială.

Verificarea ipotezelor H_A, H_B, H_{AB} se face utilizînd tabelul analizei dispersionale expus în figura 19. Înainte de a arăta modul de folosire a acelui tabel, vom preciza unele notații.

Fiind date observațiile Y_{ijk}, vom nota

$$Y_{ij*} = \frac{1}{K} \sum_{k=1}^{K} Y_{ijk}.$$

În general, semnul $*$ va indica să se calculeze o medie în raport cu acel indice; Y_{***} înseamnă, de exemplu, media aritmetică a tuturor observațiilor.

După cum se știe (vezi [11]), estimațiile parametrilor sînt date de formulele

$$\begin{aligned} &\hat{\eta}_{ij} = y_{ij*}, \quad \hat{\mu} = y_{***}, \quad \hat{\alpha}_i = y_{i**} - y_{***}, \\ &\hat{\beta}_j = y_{*j*}, \quad \hat{\gamma}_{ij} = y_{ij*} - y_{i**} - y_{*j*} + y_{***} \end{aligned} \tag{36}$$

Testarea ipotezelor se face cu ajutorul testului F. Statisticile F din tabel apar ca rapoarte de medii pătratice. O medie pătratică apare ca o sumă de pătrate împărțită la numărul său de grade de libertate.

Să considerăm, de exemplu, statistica F_A definită de

$$F_A = \frac{\dfrac{SS_A}{\nu_A}}{\dfrac{SS_e}{\nu_e}}.$$

Scanned with OKEN Scanner

Dacă ipoteza H_A este adevărată, atunci F_A este o variabilă F cu ν_A și ν_e grade de libertate, adică $F_A = F_{\nu_A, \nu_e}$. Dacă α este un prag de semnificație dat (de exemplu, $\alpha = 0.05$) și din tabelul repartiției F se extrage $F_{\nu_A, \nu_e; \alpha}$, astfel încît

$$P(F_{\nu_A, \nu_e} > F_{\nu_A, \nu_e; \alpha}) = \alpha,$$

atunci ipoteza H_A se respinge cînd $F_A \geqslant F_{\nu_A, \nu_e; \alpha}$ și se acceptă în caz contrar.

În mod similar se verifică și ipotezele H_B și H_{AB}.

Din punct de vedere practic, dificultatea constă în calculul sumelor SS ale pătratelor abaterilor. Se poate simplifica mult acest calcul dacă se utilizează formulele

$$SS_A = JK \sum_i y_{i**}^2 - IJKy_{***}^2$$

$$SS_B = IK \sum_j y_{*j*}^2 - IJKy_{***}^2$$

(37)
$$SS_{cel} = K \sum_{i,j} y_{ij*}^2 - IJKy_{**}^2$$

$$SS_{AB} = SS_{cel} - SS_A - SS_B = K \sum_{i,j} y_{ij*}^2 - JK \sum_i y_{i**}^2 -$$

$$- IK \sum_j y_{*j*}^2 + IJKy_{***}^2. \quad SS_e = \sum_{i,j,k} y_{ijk}^2 - K \sum_{i,j} y_{ij*}^2.$$

Prin utilizarea formulelor (37) se evită memorarea *abaterilor* observațiilor, cerută de formulele tabelului analizei dispersionale din figura 19.

Rezumînd, verificarea ipotezelor (35) se poate face pe baza următorului algoritm de calcul.

Pasul 0. Intrare: I, J, K, FA, FB, FAB; HA, HB, HAB sînt variabile logice de ieșire.

Alocare de memorie: vector X de dimensiune K; vector YI de dimensiune I; vector YJ de dimensiune J; matrice YIJ de dimensiune $I \times J$. (Aceștia corespund respectiv la Y_{ijk}, Y_{i**}, Y_{*j*}, Y_{ij*}).

Inițializări: $S = 0$, $YI(II) = 0$, $YJ(JJ) = 0$, $YIJ(II, JJ) = 0$, $1 \leqslant II \leqslant I$, $1 \leqslant JJ \leqslant J$.

Pasul 1. Pentru fiecare II, JJ fixați citește vectorul X de dimensiune K și calculează suma $YIJ(II, JJ)$; calculează suma S a pătratelor componentelor lui X.

Scanned with OKEN Scanner

Sursa de dispersie	SS	Numărul gradelor de libertate	$\overline{SS}$	
Linia	$SS_A = JK\sum_i (Y_{i***} - Y_{***})^2$	$\nu_A = I - 1$	$\overline{SS}_A = \frac{SS_A}{I-1}$	$F_A = \frac{\overline{SS}_A}{SS_e}$
Coloana	$SS_B = IK\sum_j (Y_{*j*} - Y_{***})^2$	$\nu_B = J - 1$	$\overline{SS}_B = \frac{SS_B}{J-1}$	$F_B = \frac{\overline{SS}_B}{SS_e}$
Interacţiunea	$SS_{AB} = K\sum_{i,j}(Y_{ij*} - Y_{i**} - Y_{*j*} + Y_{***})^2$	$\nu_{AB} = (I-1)(J-1)$	$\overline{SS}_{AB} = \frac{SS_{AB}}{(I-1)(J-1)}$	$F_{AB} = \frac{\overline{SS}_{AB}}{SS_e}$
Dispersia reziduală	$SS_e = \sum_{i,j,k}(Y_{ijk} - Y_{ij*})^2$	$\nu_e = IJ(K-1)$	$\overline{SS}_e = \frac{SS_e}{IJ(K-1)}$	—
Suma totală a pătratelor abaterilor	$SS_t = \sum_{i,j,k}(Y_{ijk} - Y_{***})^2$	$IJK - 1$		$F\nu_A, \nu_e, \alpha =$ $F\nu_B, \nu_e, \alpha =$ $F\nu_{AB}, \nu_e, \alpha =$

Fig. 19. Schema analizei dispersionale bifactoriale.

Pasul 2. Însumînd în mod corespunzător liniile şi coloanele matricei YIJ, calculează sumele $YI(II)$ şi $YJ(JJ)$ şi suma Y a elementelor lui YIJ.

Pasul 3. Divide sumele YI, YJ, YIJ, Y respectiv la $J \times K$, $I \times K$, K, $I \times J \times K$ şi depune mediile obţinute la aceleaşi locaţii.

Pasul 4. Calculează SSA, SSB, $SSAB$, $S(= SS_e)$ conform formulelor (37).

Pasul 5. Calculează statisticile F_1, F_2, F_{12} (care sînt respectiv F_A, F_B, F_{AB} din figura 19).

Pasul 6. Aplică testul F şi atribuie valorile corespunzătoare variabilelor logice HA, HB, HAB.

Programul FORTRAN pentru algoritmul precedent este expus în figura 20. Programul conţine şi instrucţiunile de intrare/ieşire şi s-a utilizat la realizarea unei aplicaţii referitoare la calitatea rulmenţilor; nivelele factorului A sînt durate de „vechime" faţă de data fabricării, iar nivelele lui B sînt diverse tratamente aplicate rulmenţilor. Din

Scanned with OKEN Scanner

analiza efectuată, s-a găsit că influenţa cea mai mare o are durata de „călire" a rulmenţilor, iar tratamentele aplicate nu produc variaţii sensibile ale calităţii.

```
      SUBROUTINE ANOVA2 (I, J, K, FA, FB, FAB, X, YI, YJ, YIJ, HA, HB,
     — HAB, SA, SB, SAB, S, Y)
      DIMENSION X(K), YI(I), YJ(J), YIJ(I, J)
      LOGICAL HA, HB, HAB
C     SE TESTEAZĂ DACA FACTORII A, B INFLUENŢEAZA VALORILE
C     OBSERVATE X. HAB ESTE IPOTEZA STATISTICA REFERITOARE
C     LA INTERACTIUNI HA ŞI HB SE REFERA RESPECTIV LA EFEC-
C     TELE FACTORILOR A ŞI B I ESTE NUMARUL NIVELELOR FAC-
C     TORULUI A IAR J NUMARUL NIVELELOR FACTORULUI B. K
C     ESTE NUMARUL (CONSTANT) AL OBSERVATIILOR DIN CELULA
C     (I, J). FA, FB, FAB SINT LIMITELE TEORETICE ALE STATISTICII
C     F. YI, YJ SINT MEDII CORESPUNZIND DIFERITELOR NIVELE ALE
C     FACTORILOR, IAR YIJ SINT MEDII PE CELULE. Y ESTE MEDIA GE-
C     NERALA. SA, SB, SAB SINT SUME DE PATRATE ALE ABATERILOR.
C     I, J, K, FA, FB, FAB, X(K) SINT PARAMETRI FORMALI DE INTRARE
C     X SINT CELE K DATE DINTR-O UNITATE DE INREGISTRARE
      S = 0.
      DO 5 II = 1, I
      DO 5 JJ = 1, J
      YIJ (II, JJ) = 0.
      READ (1, 1) (X(L), L = 1, K)
    1 FORMAT (10F5.1)
      DO 5 KK = 1, K
      YIJ(II, JJ) = YIJ(II, JJ) + X(KK)
    5 S = S + X(KK)*X(KK)
      DO 6 II = 1, I
      YI(II) = 0.
      DO 6 JJ = 1, J
    6 YI(II) = YI(II) + YIJ(II, JJ)
      DO 7 JJ = 1, J
      YJ (JJ) = 0.
      DO 7 II = 1, I
    7 YJ(JJ) = YJ(JJ) + YIJ(II, JJ)
```

Fi

Scanned with OKEN Scanner

analiza efectuată, s-a găsit că influența cea mai mare o are durata de „călire" a rulmenților, iar tratamentele aplicate nu produc variații sensibile ale calității.

```
      SUBROUTINE ANOVA2 (I, J, K, FA, FB, FAB, X, YI, YJ, YIJ, HA, HB,
     — HAB, SA, SB, SAB, S, Y)
      DIMENSION X(K), YI(I), YJ(J), YIJ(I, J)
      LOGICAL HA, HB, HAB
C     SE TESTEAZĂ DACA FACTORII A, B INFLUENȚEAZA VALORILE
C     OBSERVATE X. HAB ESTE IPOTEZA STATISTICA REFERITOARE
C     LA INTERACTIUNI HA ȘI HB SE REFERA RESPECTIV LA EFEC-
C     TELE FACTORILOR A ȘI B I ESTE NUMARUL NIVELELOR FAC-
C     TORULUI A IAR J NUMARUL NIVELELOR FACTORULUI B. K
C     ESTE NUMARUL (CONSTANT) AL OBSERVATIILOR DIN CELULA
C     (I, J). FA, FB, FAB SINT LIMITELE TEORETICE ALE STATISTICII
C     F. YI, YJ SINT MEDII CORESPUNZIND DIFERITELOR NIVELE ALE
C     FACTORILOR, IAR YIJ SINT MEDII PE CELULE. Y ESTE MEDIA GE-
C     NERALA. SA, SB, SAB SINT SUME DE PATRATE ALE ABATERILOR.
C     I, J, K, FA, FB, FAB, X(K) SINT PARAMETRI FORMALI DE INTRARE
C     X SINT CELE K DATE DINTR-O UNITATE DE INREGISTRARE
      S = 0.
      DO 5 II = 1, I
      DO 5 JJ = 1, J
      YIJ (II, JJ) = 0.
      READ (1, 1) (X(L), L = 1, K)
    1 FORMAT (10F5.1)
      DO 5 KK = 1, K
      YIJ(II, JJ) = YIJ(II, JJ) + X(KK)
    5 S = S + X(KK)*X(KK)
      DO 6 II = 1, I
      YI(II) = 0.
      DO 6 JJ = 1, J
    6 YI(II) = YI(II) + YIJ(II, JJ)
      DO 7 JJ = 1, J
      YJ (JJ) = 0.
      DO 7 II = 1, I
    7 YJ(JJ) = YJ(JJ) + YIJ(II, JJ)
```

Fig. 20

Scanned with OKEN Scanner

```
      Y = 0.
      DO 8 II = 1, I
    8 Y = Y + YI(II)
      Y = Y/(K*I*J)
      SA = 0.
      SB = 0.
      SAB = 0.
      DO 14 II = 1, I
      DO 14 JJ = 1, J
      YIJ(II, JJ) = YIJ(II, JJ)/K
   14 SAB = SAB + YIJ(II, JJ)*YIJ(II, JJ)
      DO 15 II = 1, I
      YI(II) = YI(II)/(K*J)
   15 SA = SA + YI(II)*YI(II)
      DO 16 JJ = 1, J
      YJ(JJ) = YJ(JJ)/(K*I)
   16 SB = SB + YJ(JJ)*YJ(JJ)
      S = S - SAB*K
      SAB = SAB*K - SA*J*K - SB*I*K + Y*Y*I*J*K
      SA = SA*J*K - Y*Y*I*J*K
      SB = SB*I*K - Y*Y*I*J*K
      F12 = SAB*I*J*(K - 1.)/(S*(I - 1.)*(J - 1.))
      IF(F12 - FAB)17, 18, 18
   17 HAB = .TRUE.
      GO TO 20
   18 HAB = .FALSE.
   20 F1 = SA*I*J*(K - 1.)/((I - 1.)*S)
      IF(F1 - FA)21, 22, 22
   21 HA = . TRUE.
      GO TO 24
   22 HA = .FALSE.
   24 F2 = SB*I*J*(K - 1.)/((J - 1.)*S)
      IF(F2 - FB)25, 26, 26
   25 HB = .TRUE.
      GO TO 100
   26 HB = .FALSE.
  100 CONTINUE
      RETURN
      END
```

Fig. 20 (continuare)

Scanned with OKEN Scanner

5.4. ALCĂTUIREA AUTOMATĂ A LISTELOR DE AUDIENȚĂ

5.4.1. Formularea problemei

Problema care urmează este un exemplu didactic de prelucrare de date nenumerice.

Pentru intrarea în audiență la o instituție, solicitanții se înscriu la un birou în ordinea sosirii și optează pentru a merge în audiență la unul din conducătorii X, Y, Z ai instituției, menționînd și durata T a audienței.

Alcătuirea listelor de audiență se realizează astfel:

a) Dacă solicitantul provine din instituție, el poate să meargă în audiență la conducătorul pentru care a optat.

b) Dacă solicitantul provine din afara instituției, el este inclus pe listele de audiență conform următoarelor reguli:

— dacă solicită audiență la Z, el va fi inclus fără restricții pe lista de audiență a lui Z;

— dacă solicită audiență la Y, el va fi inclus pe lista de audiență a lui Y dacă și numai dacă a mai fost în audiență la Z sau solicită timp de audiență de cel mult cinci minute. În caz contrar, el va fi inclus pe lista de audiență a lui Z;

— dacă solicită audiență la X, el va fi inclus pe lista lui X numai în cazul cînd timpul de audiență este de cel mult cinci minute și solicitantul a mai fost în audiență la unul din conducători. În caz contrar, solicitantul va fi inclus pe lista de audiență a lui Y.

Să se alcătuiească programul FORTRAN pentru tipărirea listelor de audiență pentru fiecare conducător ținîndu-se seama de ordinea solicitărilor.

5.4.2. Analiza problemei

Datele de intrare se vor organiza într-un fișier pe cartele, astfel încît fiecare solicitant să fie înregistrat pe o cartelă. Variabilele alese pentru programul FORTRAN sînt următoarele:

$A(5)$ — un masiv de cinci variabile reale pentru înregistrarea a 20 caractere alfanumerice care desemnează numele și prenumele solicitantului. (Fiecare element al masivului memorează patru caractere).

T — o variabilă reală ce reprezintă durata audienței (în minute). Presupunem că T are partea întreagă de maximum două cifre, iar partea zecimală de maximum o cifră.

Scanned with OKEN Scanner

ICOD — o variabilă numerică întreagă ce reprezintă conducătorul optat pentru audienţă şi ea ia valorile

$$ICOD = \begin{cases} 1, \text{ dacă se cere audienţă la } X; \\ 2, \text{ dacă se cere audienţă la } Y; \\ 3, \text{ dacă se cere audienţă la } Z. \end{cases}$$

IFOST — o variabilă întreagă ce descrie faptul dacă solicitantul a mai fost în audienţă sau nu şi ea ia valorile

$$IFOST = \begin{cases} 0, \text{ dacă nu a mai fost în audienţă}; \\ 1, \text{ dacă a mai fost în audienţă la } X; \\ 2, \text{ dacă a mai fost în audienţă la } Y. \\ 3, \text{ dacă a mai fost în audienţă la } Z. \end{cases}$$

I — o variabilă întreagă care descrie proveniența solicitantului şi ea ia valorile

$$I = \begin{cases} 0, \text{ dacă solicitantul este din instituţie}; \\ 1, \text{ dacă solicitantul este din afară}. \end{cases}$$

Astfel codificate datele, înregistrarea lor pe cartelă este prezentată în figura 21.

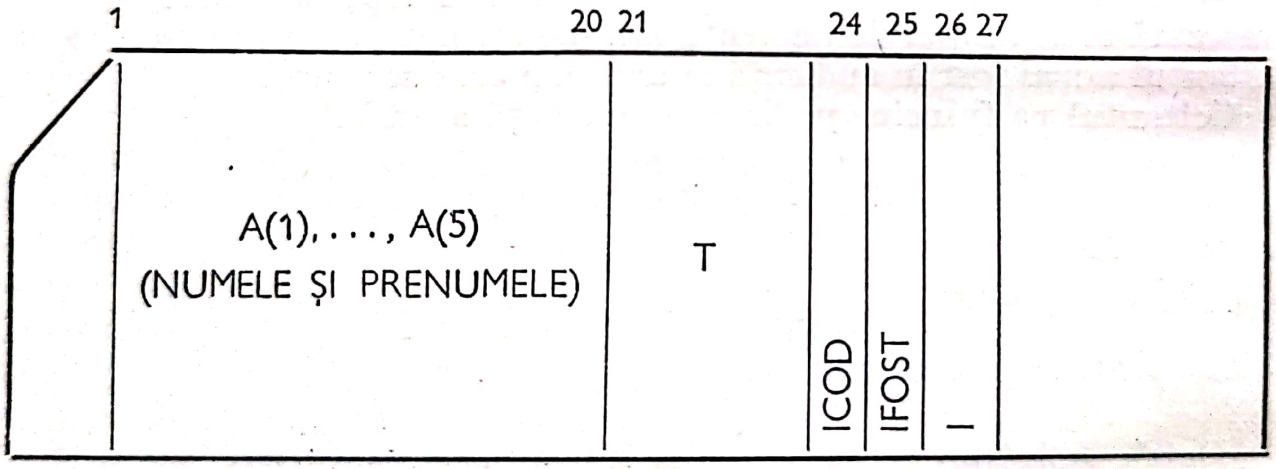


Fig. 21. Descrierea cartelei cu datele de intrare ale programului

Se presupune că ordinea cartelelor de date corespunde ordinei de solicitare a audienţelor. (Eventual, se poate include pe cartelă, în coloanele rămase libere, numărul de ordine al solicitării).

Pentru alcătuirea listelor de audienţă la cei trei conducători *X*, *Y*, *Z* se vor folosi trei masive bidimensionale *B*, *C*, *D*, astfel încît primul

Scanned with OKEN Scanner

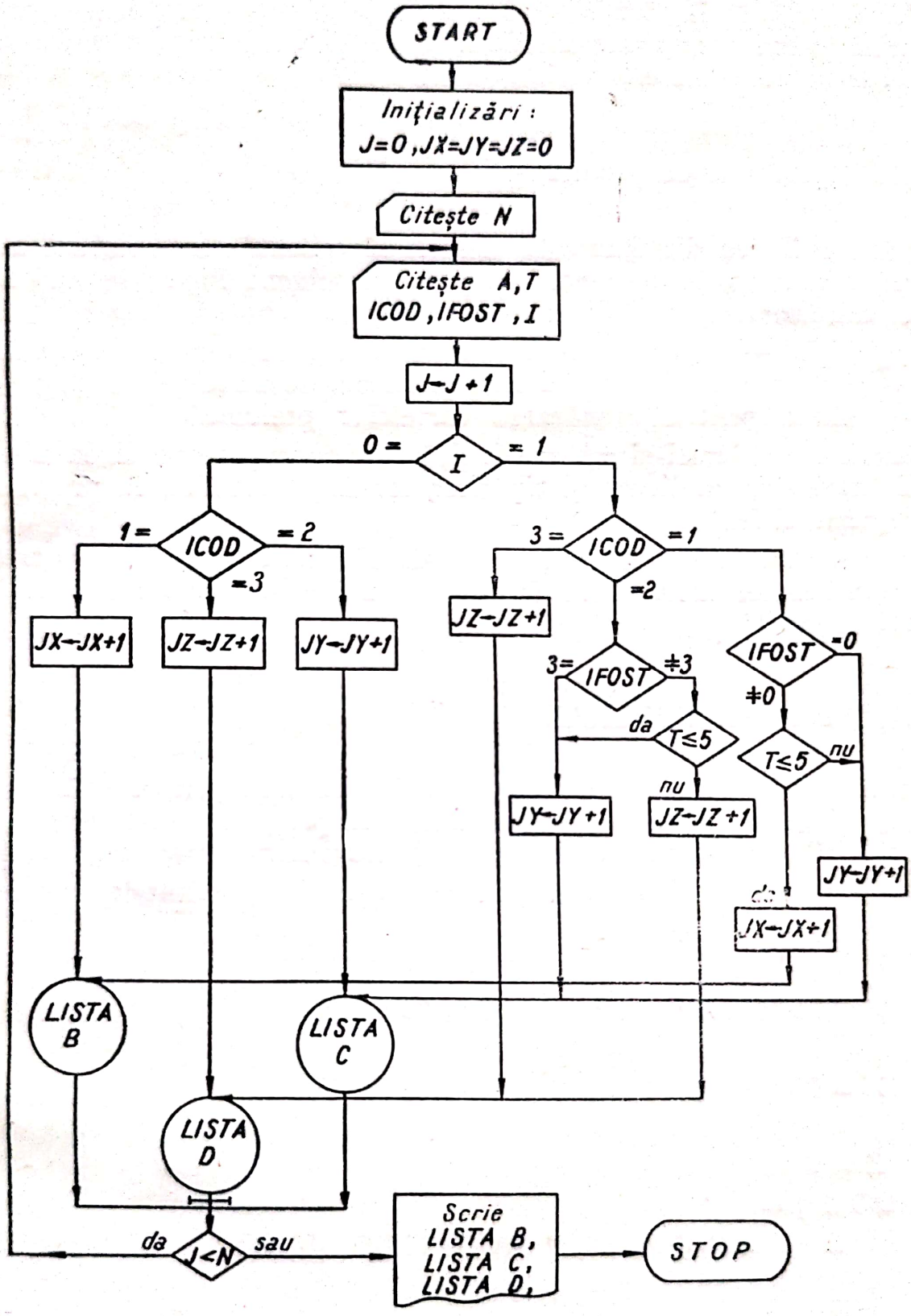


Fig. 22. Schema logică pentru alcătuirea listelor de audiențe.

Scanned with OKEN Scanner

indice să ia valorile 1,2, . . . , 5, iar cel de al doilea indice să ia valori maxime permise de limitările de memorie şi de dimensiunea fişierului de date ce urmează a fi prelucrate.

5.4.3. Algoritmul de prelucrare

Schema logică din figura 22 descrie algoritmul care conduce la alcătuirea listelor de audienţe. În plus, în schema logică se mai folosesc următoarele variabile:

JX, *JY*, *JZ* — contori pentru numărarea audienţelor din fiecare listă;
N — numărul total de solicitanţi (numărul de cartele de date);
J — contor pentru numărarea cartelelor prelucrate.

Analiza problemei şi schema logică sînt concepute pe ideea memorării listelor de audienţe şi tipărirea lor după terminarea prelucrării întregului fişier de date.

În cazul unui fişier mare de date, algoritmul de prelucrare ar trebui conceput astfel încît să se folosească memoriile externe (discuri şi benzi magnetice).

5.4.4. Programul FORTRAN

În figura 23 se prezintă programul FORTRAN pentru prelucrarea datelor şi alcătuirea listelor de audienţe. În program se foloseşte codul *FORMAT A* pentru manipularea de caractere. Programul a fost testat avînd datele de intrare din figura 23 b), iar rezultatele prelucrării sînt expuse în figura 23 c).

```
      DIMENSION A(5),  B(5, 10), C(5, 10), D(5, 10)
      N = 6
      JX = 0
      JY = 0
      JZ = 0
      DO 1 J = 1, N
      READ (1, 2) (A(II), II = 1, 5), T, ICOD, IFOST, I
    2 FORMAT (5A4, 3XF4.1,3(5XI1))
      IF(I)3, 4, 5
    3 WRITE (3, 6) J
```


Scanned with OKEN Scanner

```
 6 FORMAT (1X, I2, 3X' EROARE')
   GO TO 1
 4 IF(ICOD = 2)7, 8, 9
 7 JX = JX + 1
   DO 32 II = 1,5
32 B(II, JX) = A(II)
   GO TO 1
 8 JY = JY + 1
   DO 42 II = 1,5
42 O(II, JY) = A(II)
   GO TO 1
 9 JZ = JZ + 1
   DO 52 II = 1, 5
52 D(II, JZ) = A(II)
   GO TO 1
 5 IF(ICOD = 2)12, 14, 9
14 IF(IFOST — 3)15, 8, 15
15 IF(T — 5.)8, 8, 9
13 IF(IFOST)3, 8, 16
16 IF(T = 5.)7, 7, 8
 1 CONTINUE
   WRITE (3, 97)
97 FORMAT(//5X, ' AUDIENTE LA COND. X'/5X20(' = ')/)
   DO 62 LL = 1, JX
   WRITE (3,63) LL, (B(II, LL), II = 1,5)
63 FORMAT (' ', I 3, 3 X 5A4)
62 CONTINUE
   WRITE (3, 98)
98 FORMAT (//5X, 'AUDIENTE LA COND. Y'/5 X 20(' = ')/)
   DO 72 LL = 1, JY
   WRITE (3, 63) LL, (C(II, LL), II = 1, 5)
72 CONTINUE
   WRITE (3, 99)
99 FORMAT (//5X, 'AUDIENTE LA COND. Z'/5 X 20(' = ')/)
   DO 82 LL = 1, JZ
   WRITE (3, 63) LL, (D(II, LL), II = 1, 5)
82 CONTINUE
   STOP
   END
```

Fig. 23. a) Programul FORTRAN pentru alcătuirea listelor de audienţă

Scanned with OKEN Scanner

NUME SI PRENUME	TIMP T	COND ICOD	FOST IFOST	PROVEN I
VASILESCU C. ION	5.0	2	0	0
CONSTANTINESCU V.	3.0	1	3	1
IRIMESCU C. D-TRU	9.0	3	1	0
VASILACHE D. GEORGE	7.0	2	0	1
DRAGHICIU D. NICU	11.0	1	1	1
ENESCU H. ION	15.0	1	0	0

Fig. 23. b) Datele de intrare ale programului

AUDIENTE LA COND. X

1 CONSTANTINESCU V.
2 ENESCU H. ION

AUDIENTE LA COND. Y

1 VASILESCU C. ION
2 DRAGHICIU D. NICU

AUDIENTE LA COND. Z

1 IRIMESCU C. D-TRU
2 VASILACHE D. GEORGE

Fig. 23. c) Listele de audiențe produse de calculator

BIBLIOGRAFIE

[1] CHURCHMAN, C. W., *An analysis of the concept of simulation*, în Symposium on Monte Carlo Methods, editori Austin C. Hoggatt, Frederick E. Balderston-Cincinati, South-Western Publishing Co., 1963.

[2] HAMMERSLEY, J. M., HANDSCOMB, D. C., *Monte Carlo Methods*, Metheuen & Co, London, 1964.

[3] IBM BELGIUM, *Linear Programming Code, DOS/360*, 15.2.7 01.

[4] Linear Programming System/360, *Program* Description Manual, 360A—CO—18X, H20—0607—0.

Scanned with OKEN Scanner

[5] Mathematical Programming System/360, — *Linear and Separable Programming, User's Manual, Version II*, 360A—CO—14X, H20—0476—2.
[6] NAYLOR, TH. H., BALINTFY, J. L., BURDICK, D. S., CHU, KONG, *Computer Simulation Techniques*, John Wiley Sons, Inc. New York, London, Sydney, 1966.
[7] SHUBIK, M., *Simulation of the industry and the firm*, American Economic Revieu, L, No. 5, pp. 908—919.
[8] VĂDUVA, I., *Suprafețe de răspuns*, Studii de statistică. Lucrările celei de a doua consfătuiri științifice de statistică, București, 19 nov- 1 dec, 1963.
[9] VĂDUVA, I., *Computer Generation of Random Variables for Queueing Problems. Illustrated by a Machine Interference Problem*, M. Sc. Thesis, University of Manchester, Institute of Science and Technology, October, 1969.
[10] VĂDUVA, I., *Simulation of the Machine Interference Problem*, Bull, Math, Tome 13(61), Nr. 4, 1969.
[11] VĂDUVA I., *Analiză dispersională, București*, Ed. Tehnică 1970.
[12] VĂDUVA I., *Computer Generation of Random Variables and Vectors Related to PERT Problems*. The Fourth Conference on Probability Theory Brașov, September 1971 (sub tipar).
[13] CIOBANU GH., VĂDUVA, I., *Aplicarea simulării digitale în rezolvarea problemelor PERT cu resurse*. Analele Universității București, nr. 1. 1972 (sub tipar)
[14] WILKS, S., *Mathematical Statistics*. Princeton Univ. Press, 1962.

Scanned with OKEN Scanner

CUPRINS

Prefață 2

Capitolul I 5

INTRODUCERE ÎN CUNOAȘTEREA UNUI SISTEM DE CALCUL 5

1.1 *Introducere* 5

1.2 *Părțile componente ale unui sistem de calcul. Purtători de informație* 6

1.2.1 Părțile componente ale unui sistem de calcul 6

1.2.2 Descrierea părților componente. 7

1.2.3 Ierarhia unui sistem de calcul. Canale 9

1.2.4 Tipuri de canale 10

1.2.5 Purtători de informație 11

1.2.6 Configurația de la Centrul de Calcul al Universității din București . 15

1.3 *Baza aritmetică a unui calculator* 17

1.3.1 Sisteme de numerație. Convertirea numerelor 17

1.3.2 Tipuri de prelucrare și coduri de reprezentare a datelor numerice . . . 21

1.3.3 Structura unei instrucțiuni. Tipuri de instrucțiuni 24

1.4 *Posibilități de lucru ale unui sistem de calcul* 27

1.4.1 Posibilități fizice 27

1.4.2 Posibilități de programare ; limbaje algoritmice și compilare 30

1.5 *Despre sistemul de operare al unui sistem de calcul* 32

1.5.1 Ce este un sistem de operare și necesitatea folosirii lui 32

1.5.2 Protejarea împărțirii memoriei principale și bibliotecile rezidenței sistemului 34

1.5.3 Numele simbolice ale unităților de intrare/ieșire 37

1.5.4 Descrierea și funcțiunile unui sistem de operare concret. 39

1.5.5 Prelucrarea independentă de timp, prelucrarea în timp real, prelucrarea prin divizarea timpului, multiprogramarea și multiprelucrarea 42

1.5.6 Despre generarea sistemului de operare al unei configurații de calcul 46

1.6 *Folosirea bibliotecilor de subprograme* 47

1.6.1 Ce este o bibliotecă de subprograme a unui centru de calcul. 47

1.6.2 Descrierea folosirii unor subprograme dintr-o bibliotecă concretă . . 49

Bibliografie 52

Capitolul II 53

MODELE MATEMATICE ȘI ALGORITME PENTRU PROBLEME DE CERCETARE ȘTIINȚIFICĂ 53

2.1 *Modelul matematic și algoritmul de rezolvare al unei probleme. Ideia de calculabilitate* 53

2.1.1 Modele și algoritme 53

2.1.2 Conceptul de calculabilitate 58

2.2 *Scheme logice* 59

2.2.1 Scheme logice generale 59

2.2.2 Scheme logice orientate spre un limbaj algoritmic de programare . . . 61

2.2.3. Exemple 63

2.3 *Fazele rezolvării unei probleme cu calculatorul electronic* 65

Scanned with OKEN Scanner

2.4 *Modele liniare* 69
2.4.1 Exemple de probleme care conduc la modele liniare 69
2.4.1.1 Problema dietei 69
2.4.1.2 Organizarea rațională a producției 70
2.4.1.3 Aproximarea Cebîșev a unui sistem de funcții liniare 73
2.4.1.4 Problema transporturilor 74
2.4.2 Algoritme pentru modele liniare 75
2.4.2.1 Algoritmul simplex 75
2.4.2.2 Găsirea unei soluții posibile de bază pentru o problemă de transport 79
2.5 *Modele neliniare* 81
2.5.1 Exemple de probleme care conduc la modele neliniare 81
2.5.2 Asupra rezolvării problemelor pătratice și a problemelor neliniare 82
2.5.3 Algoritmul lui Hildreth și D'Esopo 83
2.6 *Rezolvarea numerică a ecuațiilor diferențiale ordinare* 86
2.6.1 Metoda Euler 87
2.6.2 Metoda Euler îmbunătățită 88
2.6.3 Metoda de anticipare-corectare a lui Euler 89
2.6.4 Metoda de anticipare-corectare îmbunătățită a lui Euler 90
2.6.5 Metoda de anticipare-corectare a lui Adams și Bashforth 90
2.6.6 Metoda de anticipare-corectare a lui Adams și Bashforth îmbunătățită 91
2.6.7 Metoda lui Runge-Kutta 91
Bibliografie 92
Capitolul III 94
INTRODUCERE ÎN LIMBAJUL FORTRAN 94
3.1 *Caracteristici ale limbajelor de programare, orientate pentru cercetarea științifică* 94
3.2 *Descrierea cartelei și mulțimea de caractere* 95
3.3 *Constante și variabile. Identificatori și declarări de tipuri* 97
3.4 *Expresii aritmetice și logice* 101
3.4.1 Expresii aritmetice 101
3.4.2 Expresii logice 103
3.4.3 Instrucțiuni de atribuire 105
3.5 *Funcții standard* 107
3.6 *Transfer condiționat și necondiționat* 110
3.6.1. Transfer necondiționat 110
3.6.2. Transfer condiționat 111
3.7 *Operatorul de ciclare* 116
3.8 *Instrucțiuni de intrare și ieșire* 120
3.8.1 Instrucțiunea de intrare READ cu FORMAT 120
3.8.2 Instrucțiunea WRITE cu FORMAT 121
3.8.3 Instrucțiunea FORMAT 122
3.9 *Subprogramare. Funcții și subrutine* 128
3.9.1 Funcția instrucțiune 128
3.9.2 Funcția subprogram 129
3.9.3 Subrutina 131
3.10 *Răspunsuri la exerciții* 140
Bibliografie 148
Capitolul IV 149
FOLOSIREA MEMORIILOR EXTERNE ÎN PROBLEMELE DE CERCETARE ȘTIINȚIFICĂ 149
4.1 *Organizarea datelor unei probleme* 149

Scanned with OKEN Scanner

4.1.1 Fişierul şi elementele regăsirii informaţiei 149
4.1.2 Forma înregistrărilor logice şi organizarea conţinutului fişierului . . . 151
4.2 *Crearea şi prelucrarea fişierelor de pe discuri magnetice* 153
4.2.1 Unităţile de discuri — memorii cu acces direct. Organizarea şi prelucrarea fişierelor secvenţial şi direct 153
4.2.2 Instrucţiunile de definire şi de descriere a fişierelor pe discuri 157
4.2.3 Instrucţiunea DEFINE FILE şi pregătirea discului pentru crearea unui fişier în acces direct 160
4.2.4 Exemple de fişiere pe disc, prelucrate prin programe scrise în FORTRAN 162
4.3 *Crearea şi prelucrarea fişierelor de pe benzi magnetice* 169
4.3.1 Înregistrarea informaţiilor pe bandă. Instrucţiuni de manevrare a benzii 169
4.3.2 Instrucţiunile de definire şi de descriere a fişierelor de pe benzi . . . 172
4.3.3 Exemplu de folosire a benzii magnetice într-un program scris în FORTRAN 174
Bibliografie 176

Capitolul V 177

PROBLEME REZOLVATE 177

5.1 *Aplicarea programului* LPC *la rezolvarea unei probleme de programare liniară* 177
5.2 *Modele de simulare cu calculatorul* 185
5.2.1 Definiţia simulării, modele de simulare 185
5.2.2 Model de simulare pentru studiul endemiei tuberculozei 190
5.2.3 Model de simulare pentru problema interferenţei maşinilor 195
5.3 *Câteva probleme de statistică matematică*. 201
5.3.1 Testul χ^2 pentru verificarea normalităţii multidimensionale 202
5.3.2 Analiză dispersională bifactorială 205
5.4 *Alcătuirea automată a listelor de audienţă* 211
5.4.1. Formularea problemei 211
5.4.2. Analiza problemei 211
5.4.3. Algoritmul de prelucrare 214
5.4.4. Programul FORTRAN 214

Bibliografie 216

Scanned with OKEN Scanner

Coli de tipar: 14

Tiraj: 4 300+120 ex. leg.
Apărut: 1973

Întreprinderea Poligrafică Cluj
Str. Brassai nr. 5—7
Republica Socialistă România

Comanda nr. 263/5204

Scanned with OKEN Scanner